
HAPPY C64

* Documentation de Happy C64 *

Version SDK :	0.2.3.5
Update de la doc:	22/04/2021
Programmeur :	Loïc Lété
Remerciement :	Eric Boez,Lionel Paul, Olivier Cappelar,Eric Cottencin, ma chérie.

* Caractéristiques du C64 *

Tableau des caractéristiques du Commodore 64	
CPU	Mos 6510 0,985 MHz (pal) / 1,023 MHz(NTSC)
Mémoire Ram	64ko
Mémoire Rom	20 ko (8ko basic, 8ko kernal, 4ko tiles pré-généré)
Résolution d'affichage	320x200 pixel sans le border.
Résolution d'un pattern	8x8 pixel.
Nombre de pattern à l'écran	40x25 pattern (1000)
Nombre de sprites machine	8
Taille d'un sprite	24x21 pixel
Fonction sur les sprites	Doubler la largeur, doubler la hauteur, détection de collision avec un autre sprite / un tile.
Scrolling Hardware	Horizontal / Verticale sur une amplitude de 8 pixels
Joystick	2 ports avec contrôle Haut/BAS/GAUCHE/DROITE et bouton feu
Musique	Trois voix / 8 octaves / Triangle,Dents de scie,carré/bruit
Nuancier	16 couleurs

==========
* Vocabulaire *
==========

Pattern	C'est tout simplement l'encodage en octet d'un élément graphique. (Tiles ou Sprite). Un Tiles carré peut être encodé avec les valeurs binaire suivant ! 0B11111111,0b10000001,0b10000001,0b10000001,0b10000001,0b10000001,0b10000001,0B11111111
Tiles	Un tile c'est un élément graphique affichable sur l'écran dans un quadrillage. Il peut représenter une police de caractère, (texte), ou des éléments du décore. On peux utiliser le terme de character , tuile ou caractère.
Tilemap	C'est la représentation en mémoire de l'écran. Elle se situe dans la mémoire écran (screen mémorie) en organisation linéaire. Le premier octet de la mémoire écran c'est la position 0,0 (en case) et la valeur contenu dans cette octet représente l'index du tile à afficher à cette endroit.(La taille de la tilemap fait 1000 octets)
Sprite	Ou Lutin, MOB (Movable Object Block), BOB, sont des éléments graphiques qui peuvent être placé au pixel près sans effacé ce qui se trouve à l'arrière. Le C64 gère 8 sprites simultanément de 24x21 points. (ou 12x21 points logique doublé sur la largeur en mode multicolore).
Numéros de BIT	On parle souvent de numéros de bit dans le document. C'est la place du bit dans l'octet. Sur 1 octet on numérote les bits de 0 à 7 et on compte de droite à gauche. Exemple %00100000 dans cette exemple le bit à 1 c'est le 5em bit de l'octet. En langage C, une valeur binaire s'écrit avec un préfixe 0b dans la documentation, c'est le sigle % qui se trouve au début d'une valeur binaire.
Registres	Un registre est une adresse mémoire dans la Ram du C64 qui permet de contrôler un composant de celui si. Par exemple le registre $0 permet d'activer ou non la Rom du Basic, du Kernal...

Table des matières

* Présentation *

* Introduction *

Happy C64 est une petite bibliothèque pour programmer le commodore 64 avec le langage de programmation C et le compilateur CC64.

L'esprit de la bibliothèque se veut être proche du système avec une paire de fonction évolué pour simplifier le développement de votre programme. (Module de texte, gestion des tiles/couleurs…) Mais n'a pas pour but de faire le café à votre place. Une connaissance de la machine est quand même primordiale et la connaissance du langage C est obligatoire car ce guide ne vous l'apprendra pas.

* Mise en place de votre projet *

1 : Les outils

Pour programmer le commodore c64 en langage C en cross plateforme, il vous faut des outils de programmation.

- Un émulateur C64 qui permet de tester votre programme sur votre PC. VICE est sympathique. Personnellement j'utilise la suite d'émulateur de **Cloanto** : C64 Forever.

- Le compilateur CC65 qui permet de compiler votre programme. Le Git possède une version CC65 qui fonctionne mais n'est pas à jour. Je vous conseille de le re télécharger.

- Un éditeur de texte pour écrire vos programmes. En vrac, notepad++, Vscode,notepad…

2 : l'organisation du dossier de l'api

Ceci est un exemple d'organisation de dossier qui fonctionne directement avec mon fichier bat. Si vous avez de l'expérience dans ce domaine, vous pouvez vous confectionner un fichier makefile et chambouler l'organisation des dossier pour adapter tout ça à votre convenance. Si vous débutez, garder cette organisation le temps de vous faire la main.

Il y a quatre dossiers majeur pour happyc64.
Le dossier principale HappyC64 qui va regrouper les trois dossier majeur.

- Le dossier **CC65** qui va contenir le compilateur CC65. Placer directement le contenu de cc65 : les dossier "bin","cfg","include"…

- Le dossier **hpc** qui va contenir la librairie de happyc64.

- Le dossier **projets** qui va contenir vos projets.

- Le dossier **outils** qui contient les outils divers de happyc64

Le dossier **hpc** contient un dossier **header** avec *happyc64.h* dedans et deux autre fichier. *Happyc64.c* qui est le code source de la librairie (non utile), et *happyc64.lib* qui est la librairie de l'API en elle même.

Le dossier **outils** contient deux sous dossiers, **cc1541** qui est le logiciel de création de disquette, **makefile** qui contient le logiciel make.exe , un fichier makefile et .bat à copier à la racine du dossier de vos projets.

3 : organisation du dossier de votre projet

C'est dans le dossier projets que vous aller déposer vos créations. Un dossier par création.

L'organisation de votre dossier de jeu se fait en deux dossiers majeurs , le fichier make.bat et makefile.

Le premier dossier se nomme **bin**. Il va accueillir vos fichier.prg, fichier.d64 ou fichier.tap en fonction de ce que vous aller faire.

Le dossier **source** est la pour votre code source du jeu.

En fonction de vos besoin d'autre dossier peut être crée comme par exemple un dossier data pour vos fichier data.sql ou autre…

* Fichier de compilation *

note : *HappyC64 contient un makefile dans le dossier outils pour mieux automatiser la compilation. Le fichier compilation.bat devient pratiquement inutile.*

Nous allons voir le fichier de compilation.bat. C'est un fichier exécutable qui va envoyer des ordres à votre ordinateur pc windows. Il va permettre de compiler votre programme, de créer le fichier.prg par exemple, de transformer ça en .d64, et pourquoi pas de lancer le programme automatique avec vice.

CC65 doit convertir votre programme écrit en C, dans le langage du commodore 64. Les étapes simplifiés sont : votre code source en C se transforme en Assembleur. Le code Assembleur en langage machine . Puis réunir tous les fichiers transformé en langage machine pour en faire un fichier .prg. C'est le rôle de CC65

Voici un simple exemple de fichier de compilation.bat.

Code : Fichier compilation.bat

```
echo off
rem ----------------------------------------
rem * Configuration du fichier de compilation *
rem ----------------------------------------

set cl65=..\..\cc65\bin\cl65
set cc65=..\..\cc65\bin\
set adr_source=source\
set adr_out=bin
set prg_prog=c64.prg
set adr_happy=..\..\hpc

set fichier= %adr_source%main.c

rem -------------------------------------
rem * preparation du dossier exportation *
rem -------------------------------------
if exist "%adr_out%\*.prg" del %adr_out%\*.prg
if exist "%adr_out%\*.d64" del %adr_out%\*.d64
if exist "%adr_out%\*.t64" del %adr_out%\*.t64
if exist "%adr_out%\*.rp9" del %adr_out%\*.rp9

rem ---------------
rem * compilation *
rem ---------------

%cl65% -o %adr_out%/%prg_prog% -u __EXEHDR__  -O -t c64  --include-dir
%adr_happy%/header %fichier% %adr_happy%\happyc64.lib
pause

rem ----------
rem * menage *
rem ----------
if exist "%adr_source%*.o" del %adr_source%*.o
pause
```

Explication :
Dans la zone configuration, vous pouvez modifier des dossiers et des liens. Normalement si vous suivez mon organisation, il y a seulement le nom du programme à changer.
Note : On CC65 exporte que des fichiers programmes, sur c64 forever, il faut lui faire croire que c'est une disquette, avec un .d64
mais ça reste finalement qu'un fichier programme.
Sur un symple vice, vous devez refaire une "vrais disquette" avec par exemple dir master.

set_fichier : Chaque fichier C de votre programme, doit être inscrit ici avec %adr_source%nom_fichier.c
CC65 ne gère pas les *c comme sur SDCC 32 bits malheureusement.
Si vous cliquez dessus des messages d'erreur de fichier prg par exemple doit apparaître la premier fois mais pas grave. Le fichier prg doit apparaître dans out. Si c'est le cas bravos vous avez compilé votre premier programme pour le C64.

Note : Il est vivement conseillé d'utiliser la norme d'encodage **ANSI** ou **UTF8** pour vos fichier. CC65 semble avoir des problèmes avec d'autre type d'encodage comme le **UTF8-Bom** par exemple.

* Le VIC II (Video) *

* Introduction *

Le **V**ideo **I**nterface **C**hip **II** est le contrôleur vidéo du commodore 64.
Il ne peut adresse que 16 ko de Ram, il doit donc être positionné dans une des 4 plages mémoire de la Ram C64. Il n'a pas sa propre mémoire comme les consoles ou certain micro ordinateur comme la gamme des MSX.

Il permet d'avoir une résolution de 320x200 points logique (ou 160x200 point logique en multi-couleur), afficher 40x25 tiles à l'écran. Mode Bitmap ou tiles, une palette de 16 couleurs, 8 sprites machines, scrolling, …

Le vic II est contrôlé par les registres compris entre ***$D000*** et ***$D3FF***.

* Configurer la plage de lecture du VIC *

Le Vic II doit être positionné dans une des 4 plages mémoire du C64. C'est dans cette plage mémoire que va se trouver la mémoire écran, la lecture des sprites et des tiles. Au démarrage du C64, le VIC II se trouve à l'adresse **$0000**

*void set_vic_bank(**unsigned char** id_bank)*	
***Unsigned char** id_bank*	*valeur comprise entre 0 et 3 (Valeur Réel du tableau ci dessous) pour avoir le choix entre 4 plages mémoire.*
Le gestion des banks du VIC II se fait par l'adresse *$DD00*. Le bit 0 et 1 contrôle la place du VIC II dans la mémoire. (Voir Valeur Reel du tableau de dessous)	

Macro des Banks de mémoire du VIC II		
#DEFINE	VALEUR REEL	PLAGE DE LECTURE DU VIC II
VIC_BANK_0	3	Adresse *$0000* -> *$3FFF*
VIC_BANK_1	2	Adresse *$4000* -> *$7FFF*
VIC_BANK_2	1	Adresse *$8000* -> *$BFFF*
VIC_BANK_3	0	Adresse *$C000* -> *$FFFF*
Note : Les numéros des banks dans les defines sont inversées par apport au numéros des banks dans les livres techniques du C64. C'est plus logique de partie en Bas de la Ram pour la Bank 0 (Adresse $0) et du haut de la Ram pour la Bank 3.($C000)		

* Activer / Désactiver l'affichage video *

Les deux fonctions jouent un rôle sur l'affichage vidéo . Activer ou non l'affichage vidéo. Quand l'affichage vidéo n'est pas activé, la surface de l'écran est remplacé par la couleurs du Cadre.(Border).

C'est le bit **4** du registre *$D011* qui contrôle l'affichage vidéo.

SCREEN_ON
Active l'affichage vidéo.

SCREEN_OFF
Désactive l'affichage vidéo. L'écran prend la couleur du Border.

Note : Les deux "fonctions" sont des défines.

* Attendre le début du Vblank *

***void** wait_vbl()*
Le Vbblank est le signale du retour de électrons de votre TV. (enfin il n'y a plus cela sur les tv moderne mais le signale existe toujours).

* Attendre un certain nombre de vbl *

***void** wait_time(unsigned char value)*
Permet d'attendre un certain nombre de retour d'écran. (0-255) (5 secondes maximum)

* Récupérer le niveau de ligne du balayage ecran *

***unsigned int** get_raster()*
Permet de connaître à qu'elle ligne se situe le faisceaux.

* Activer/Désactiver les interruptions *

Les **Interruptions** sont géré par le registre *$DC0D*

Void set_interruption_on()
Active les interruptions du Commodore 64

Void set_interruption_off()
Désactives les interruptions du Commodore 64

* Modifier l'emplacement de la mémoire écran *

Le C64 réserve 1024 octets pour stocker la tilemap(1000 octets) avec les pointeurs de sprite (8 octets). L'adresse de la mémoire écran au démarrage, se trouve à l'adresse 1024. ($0400) Il est possible de choisir un autre emplacement pour la mémoire écran indexé sur le vic.

*void Set_adresse_screen_memory(**unsigned char** screen_memory_pointeur)*	
***unsigned char** screen_memory_pointeur*	*Adresse sur Screen Memory en fonction de l'adresse vicII*
Le pointeur de la mémoire écran se trouve à l'adresse $D018 dans les 4 derniers bits. (Bits 4 à 7) ce qui fait 16 possibilités.	

#Define du SCREEN MEMORY (Adresse du VIC2 + adresse du screen memory)		
#DEFINE	REEL	ADRESSE OFFSET
SM_0	0	Adresse $0000
SM_0400	16	Adresse $0400
SM_0800	32	Adresse $0800
SM_0C00	48	Adresse $0C00
SM_1000	64	Adresse $1000
SM_1400	80	Adresse $1400
SM_1800	96	Adresse $1800
SM_1C00	112	Adresse $1C00
SM_2000	128	Adresse $2000
SM_2400	144	Adresse $2400
SM_2800	160	Adresse $2800
SM_2C00	176	Adresse $2C00
SM_3000	192	Adresse $3000
SM_3400	208	Adresse $3400
SM_3800	224	Adresse $3800
SM_3C00	240	Adresse $3C00

Note : Ceci est un offset par apport à l'adresse du VIC.
*Exemple ; si le vic II est branché à l'adresse **$8000**. l'utilisation de SM_0400 placera le screen memory à l'adresse **$8400**.*

==========
*** Modifie le nombre de colonne à afficher ***
==========

Le Vic II permet deux affichage écran sur la largeur. 40 ou 38 colonnes.
L'utilité de passer en mode 38 colonnes est de permettre de préparer un scrolling. C'est le **Bit 3** du registre *$D016* qui contrôle cette affichage.
(1=40 colonnes. 0=38 colonnes). Au démarrage le C64 est en **40 colonnes**.

void set_38_columns()
Le VIC II affiche 38 colonnes de largeur.

void set_40_columns()
Le VIC II affiche 40 colonnes de largeur. (Par défaut)

==========
*** Modifie le nombre de ligne à afficher ***
==========

Le Vic II permet de configurer deux affichages écran sur la hauteur. 25 ou 24 lignes.
L'utilité de passer en mode 24 lignes est de préparé un scrolling.
C'est le **Bit 3** du registre *$D011* qui contrôle cette affichage.
(1=25 lignes. 0=24 lignes). Au démarrage le C64 est en **25 lignes**.

void set_24_raws()
Le VIC II affiche 24 lignes en hauteur.

void set_25_raws()
Le VIC II affiche 25 lignes en hauteur. (Par défaut)

==========
*** Scrolling ***
==========

Un scrolling c'est l'effet de déplacer "l'écran". Le commodore 64 permet de faire du scrolling verticale et horizontal.

void set_scrolling_horizontal(**signed char** Scroll_x)	
unsigned char scroll_x	Pas du scrolling Horizontal. Valeurs entre -7 et 7 (pixel)
Permet de déplacer l'écran horizontalement d'une valeur comprise entre -7 et 7 Les bits **0-2** du registre *$D016* permettent de contrôle le scrolling Horizontalement.	

void set_scrolling_vertical(**signed char** Scroll_y)	
unsigned char *scroll_y*	*Pas du scrolling verticale. Valeurs entre -7 et 7 (pixel)*
Permet de déplacer l'écran verticalement d'une valeur comprise entre -7 et 7 Les bits **0-2** du registre *$D011* permettent de contrôle le scrolling verticale.	

==========
* Modifier la couleur du border *
==========

*void set_color_border (**unsigned char** color_id)*	
***unsigned char** color_id*	Couleur à afficher dans le cadre de l'écran. (Border)
Le registre *$D020* mémorise la couleur du Border. La valeur se trouve dans les bit **0-3**. (16 possibilités)	

L'écran du Commodore 64 possède un contour d'une couleur unis. Cette couleur peut être paramétré. Notons que la couleur du border prend place sur tout l'écran quand le contrôleur vidéo est désactivé.

==========
* Modifier la couleur du Background *
==========

*void set_color_background (**unsigned char** color_id)*	
***unsigned char** color_id*	Couleur à afficher dans la partie du fond de l'écran.
Le registre *$D021* mémorise la couleur du background. La valeur se trouve dans les bit **0-3**. (16 possibilités)	

Le Fond de l'écran possède une couleur unis qui peut être différent de la couleur du border. Le fond de l'écran portent différent nom. (screen, Background, paper (sur CPC).

Un pixel d'un charset non allumé laisse passer la couleur du background.

==========
* Modifier la couleur du Background 1 *
==========

La couleur du Background 1 est utilisé dans le mode multi couleur et étendu.
Elle permet d'afficher plus de couleur dans un pattern. (Point allumé pour le mode Multi couleur, fond pour le mode étendu).
Notons que hors Raster, La couleur est commune à tous les patterns.

*void set_color_background_1 (**unsigned char** color_id)*	
***unsigned char** color_id*	Couleur à afficher pour le **background 1** en mode **multicolore** et **étendu**.
Le registre *$D022* mémorise la couleur du background 1. La valeur se trouve dans les bit **0-3**. (16 possibilités)	

==========
* Modifier la couleur du Background 2 *
==========

La couleur du Background 2 est utilisé dans le mode multi couleur et étendu.
Elle permet d'afficher plus de couleur dans un pattern. (Point allumé pour le mode Multi couleur, fond pour le mode étendu).
Notons que hors Raster, La couleur est commune à tous les patterns.

*void set_color_background_2 (**unsigned char** color_id)*

unsigned char *color_id*	Couleur à afficher pour le **background 2** en **mode multicolore** et **étendu**.
Le registre *$D023* mémorise la couleur du background 2. La valeur se trouve dans les bit **0-3**. (16 possibilités)	

==========
*** Modifier la couleur du Background 3 ***
==========

La couleur du Background 3 est utilisé dans le mode étendu.
Elle permet afficher plus de couleur dans un pattern au niveau du FOND
Notons que hors Raster, La couleur est commune à tous les patterns.

void *set_color_background_3 (**unsigned char** color_id)*	
unsigned char *color_id*	Couleur à afficher pour le **background 3** en mode **étendu** seulement.
Le registre *$D024* mémorise la couleur du background 3. La valeur se trouve dans les bit **0-3**. (16 possibilités)	

==========
*** PAL/NTSC ***
==========

unsigned char *get_system()*
*Permet de connaître si le commodore est une version NTSC (**0**) ou PAL (**1**)*

* Gestion des tiles *

* Introduction *

Les tiles sont des éléments graphiques de 8x8 pixels qui s'affiche à l'écran sur un quadrillage invisible. Différents type de nom sont données au tiles. (font,caractère, characters,tuile,mosaïque...) mais c'est belle et bien la même chose. Ils permettent de représenter les graphismes de votre jeu. (Murs, sol, porte…)
Le Commodore possède un jeu de tile en Rom, mais il est possible de créer vos propre set graphique.(Ce qui est recommandé de tout façon)
Une ligne d'un tile est contenu dans un octet. (8 bits), chaque bit dans le mode normale représente un point. (allumé ou non) et comme il y a 8 lignes, il faut donc 8 octets pour représenter les graphismes d'un tile. L'organisation des 8 octets se nomme **pattern** ! Vous pouvez aussi retrouver le terme *Générateur de caractère*.

Le c64 possède des patterns près enregistré qui permet d'afficher les lettres quand on écrit en basic. Elle se situe à l'adresse $1000. Mais il est possible de choisir un autre emplacement de la ram qui doit se situer dans la plage lisible du VIC.

* Modifier l'emplacement de lecture des tiles *

void set_location_character(unsigned char id)	
unsigned char id	ID emplacements des patterns par apport à l'adresse du VICII
Le pointeur de character se trouve dans le registre *$D018* au bit **1-3**. Ce qui donne 8 emplacements de départ pour lecture des patterns.	

Cette fonction permet de choisir l'emplacement de lecture des tiles.
Note : Tout comme le screen Memory, l'adresse est un "offset" par apport à l'adresse de départ du VIC II.

Tableau emplacement du pointeur de pattern.	
ID	Adresse du pattern 0
0	$0000
2	$0800
4	$1000
6	$1800
8	$2000
10	$2800
12	$3000
14	$3800

* Modifier l'adresse interne de la tilmap *

void set_adresse_tilemap(unsigned int adresse)	
unsigned int adresse	Adresse physique du screen memory

Permet de modifier la variable interne du sdk pour pointer la mémoire écran du C64.Cette variable est utilisé par les fonctions **draw_*character()*** *et* ***draw_full_character().***

Techniquement à chaque changement de bank du vicII et du pointeur de screen memory en passant par les fonctions du sdk, cette variable est recalculé.

Note : C'est une fonction très peux utilisé manuellement mais qui existe en cas ou.

* Transférer des patterns au bonne endroit ! *

Void set_data_character(unsigned int adr_cible, unsigned char data_character, unsigned char nb_pattern)*	
unsigned int adr_cible	Adresse Cible du transfère de data.
unsigned char data_character*	Pointeur (adresse) ou se trouve le 1er pattern.
unsigned char nb_pattern	Nombre de pattern à transférer.

Cette fonction permet de transférer des données d'un tableau à l'endroit voulu de la mémoire. Attention c'est bien le nombre de pattern à placer en arguments et non le nombre d'octet totale. (La fonction calcule de lui même cette dernière)

* Afficher un tile à l'écran *

La résolution du commodore 64 est de 320 x 200 pixel. Ce qui permet d'afficher 40 tiles en largeur et 25 tiles en hauteur.

La mémoire écran permet de mémoriser l'index du tiles à afficher à l'écran. (Entre 0 et 255). L'encodage des tiles est linéaire. (1er octet de la mémoire écran = position 0,0. 2em octet = position 1,0…

void draw_character (**unsigned char** position_x , **unsigned char** position_y , **unsigned char** id_character)	
unsigned char position_x	Position X du pattern à afficher. (0-39)
unsigned char position_y	Position Y du pattern à afficher. (0-24)
unsigned char id_character	ID du pattern à afficher. (0-255)

Cette fonction permet donc de poser un tile à l'écran en choisissant les coordonné X et Y. (En case)

* Récupérer l'id du charset *

Fonction qui permet de récupérer l'id du charset posé au coordonnée choisis dans le screen_map.

Unsigned char get_id_character (unsigned char position_x , unsigned char position_y)	
unsigned char position_x	Position X du pattern à afficher. (0-39)
unsigned char position_y	Position Y du pattern à afficher. (0-24)

* Modifier la couleur d'une case *

En mode standard l'encodage d'un tile sur C64, n'indique pas les couleurs à afficher, mais indique seulement si un point est allumé (bit 1) ou éteint (bit 0).

La couleur du groupe de point allumé est définie dans la color ram.
Notons avec ce system, c'est une couleur identique par groupe de 8x8 points.

void set_color_map (unsigned char position_x , unsigned char position_y ,*unsigned char color_id*)	
unsigned char position_x	Position X de la case. (0-39)
unsigned char position_y	Position Y de la case. (0-24)
unsigned char color_id	Couleur à afficher. (0-15)
Le début de la **color map** se trouve à l'adresse *$D800.*	

* Récupéré la couleur d'une case *

Cette fonction permet de récupéré le numéros de la couleur à une position voulu dans la color map.

Unsigned char get_color_map (unsigned char position_x , unsigned char position_y)	
unsigned char position_x	Position X de la case. (0-39)
unsigned char position_y	Position Y de la case. (0-24)

Tableau des couleurs	
#DEFINE	Index de la couleur
C_BLACK	0
C_WHITE	1
C_RED	2
C_TURQUOISE	3
C_PURPLE	4

C_GREEN	5
C_BLUE	6
C_YELLOW	7
C_ORANGE	8
C_BROWN	9
C_LIGHT_RED	10
C_GREY	11
C_GREY_2	12
C_LIGHT_GREEN	13
C_LIGHT_BLUE_2	14
C_GREY_3	15

==========
* Remplir la color ram d'une même couleur *
==========

void *cls_color_ram(**unsigned char** color)*	
***unsigned char** color*	Index de la couleur à remplir dans la color ram

Permet tout simplement de remplir en totalité la colors ram avec la couleur de votre choix.

==========
* Afficher un tile et choisir sa couleur ! *
==========

***void** draw_full_character(**unsigned char** position_x, **unsigned char** position_y, **unsigned char** id_character,**unsigned char** color_id)*		
***unsigned char** position_x*	Position X du pattern à afficher.	(0-39)
***unsigned char** position_y*	Position y du pattern à afficher.	(0-24)
***unsigned char** id_character*	ID du pattern à afficher.	(0-255)
***unsigned char** color_id*	Couleur à afficher.	(0-15)

Une fonction utile qui permet d'afficher un pattern à l'endroit voulu et de choisir la couleur de la case.

* Dupliquer un pattern horizontalement *

void draw_character_line_H(*unsigned char* px,*unsigned char* py,*unsigned char* size, *unsigned char* id_character,*unsigned char* color);	
unsigned char px	Position X de départ du pattern à afficher.
unsigned char py	Position Y de départ du pattern à afficher.
unsigned char size	Nombre de fois que le pattern va être dupliqué
unsigned char id_character	Index du pattern à dupliquer
unsigned char color	Index de la couleur du pattern à dupliquer.

Permet de dupliquer Horizontalement un pattern dans le screen memory. Utile pour tirer un trait ou créer des HUD.

* Dupliquer un pattern verticalement *

void draw_character_line_V(*unsigned char* px,*unsigned char* py,*unsigned char* size, *unsigned char* id_character,*unsigned char* color);	
unsigned char px	Position X de départ du pattern à afficher.
unsigned char py	Position Y de départ du pattern à afficher.
unsigned char size	Nombre de fois que le pattern va être dupliqué
unsigned char id_character	Index du pattern à dupliquer
unsigned char color	Index de la couleur du pattern à dupliquer.

Permet de dupliquer Verticalement un pattern dans le screen memory. Utile pour tirer un trait ou créer des HUD.

* Activer le mode Mode Multicolore *

SET_MULTICOLOR_MODE_ON
Active le mode multicolore. (C'est une DEFINE pas besoin de ())

Chaque point d'un tile peut prendre une des 4 configurations suivante en fonction de son encodage.

Encodage de bit	Couleur à afficher
%00	Couleur du Background
%01	Couleur du Background 1
%10	Couleur du Background 2
%11	Couleur contenue dans la color ram. (Mais seulement sur une valeur comprise entre 0 et 8)

Un tile peut donc afficher 4 couleurs sur une zone de 8x8 pixels.
- Trois couleurs général, et la couleur de sa case.

La pattern d'un tile est toujours encodé sur 8 octets.
En mode multicolore on peux définir que des tiles de 4x8 points, mais le pixel est doublé en largeur pour arriver à la taille de 8x8 pixels. (Double pixel en largeur.)

* Désactiver le mode Mode Multicolore *

SET_MULTICOLOR_MODE_OFF
Desactive le mode multicolore. (C'est une DEFINE pas besoin de ())

Permet de désactiver le mode multicolor et de repasser dans le mode standard.

* Activer Mode Étendu *

SET_EXTENDED_BACKGROUND_COLOR_ON
Active le mode étendu. (C'est une DEFINE pas besoin de ())

Le mode étendu permet d'afficher une des 4 couleurs au "fond" + la color ram du bloc 8x8 pixel.(et de rester en mode 8x8 pixel comme le mode standard)

Le choix de la couleur du fond se fait dans les 2 dernier bit du screen memory.

Encodage de bit	Couleur à afficher
%00xxxxxx	Couleur du Background
%01xxxxxx	Couleur du Background 1
%10xxxxxx	Couleur du Background 2
%11xxxxxx	Couleur du Background 3

Il ne reste que 6 bits pour choisir l'index du pattern à afficher (64 possibilités)

* Désactiver le Mode Étendu *

SET_EXTENDED_BACKGROUND_COLOR_OFF
Désactive le mode étendu. (C'est une DEFINE pas besoin de ())

* Le Mode Bitmap *

* Activer le Mode Bitmap *

SET_STANDARD_HIGHT_RESOLUTION_BMM_ON
Activer le mode Bitmap. (C'est une DEFINE pas besoin de ())

* Désactiver le Mode Bitmap *

SET_STANDARD_HIGHT_RESOLUTION_BMM_OFF
Désactive le mode Bitmap. (C'est une DEFINE pas besoin de ())

* Afficher un Pixel à l'écran *

Void draw_pixel(**unsigned int** px,**unsigned int** py)	
Permet d'afficher un pixel à l'écran.	
Unsigned int px	Position X du pixel
Unsigned int py	Position Y du pixel

* Modifier la couleur d'un bloc 8x8 *

Void set_bitmap_color_map(**unsigned char** px, **unsigned char** py, **unsigned char** color_ink, **unsigned char** color_background)	
Permet d'afficher un pixel à l'écran.	
Unsigned char px	Position X du bloc de couleur
Unsigned char py	Position Y du bloc de couleur
unsigned char color_ink	Couleur de l'encre du bloc
nsigned char color_background	Couleur de fond du bloc

==========
* CLS COLOR MAP BITMAP *
==========

Void cls_bitmap_color_ram(unsigned char ink_color,unsigned char background_color)	
Permet de remplir toute la color map de la couleurs de votre choix.	
unsigned char ink_color	Couleur de l'encre du bloc
nsigned char background_color	Couleur de fond du bloc

Void cls_bitmap()
Permet d'effacer la mémoire écran en mode bitmap

==========
* Configuration du mode Bitmap *
==========

Le mode Bitmap s'utilise d'une autre manière que le mode charset. Voici un petit guide.

La contrainte des couleurs reste identiques au mode charset.

Le "Screen Memory" se place dans la mémoire des patterns de charset qui se configure avec la fonction set_location_character(id).

Le "Color Screen" se trouve dans le screen memory du mode charset qui se configure avec la fonction et_adresse_screen_memory(id);

Comme le mode standard, chaque octet de la color screen représente une zone de 8x8 pixel à l'écran. Les **4 bits** du poids **fort** représente la couleur du pixel allumé, et les **4 bits** du poids **faible** représente le fond.

* Les Sprites *

Les sprites (ou lutin,MOB..) sont des graphismes qui peuvent être placé au pixel près sur l'écran. Elles ont la particularités de ne pas effacer le background. C'est le Vic qui gère les sprites. Le commodore 64 peut gérer et afficher 8 sprites à l'écran. Chaque sprite à des propriétés. (Position X, Position Y, activer ou non le sprite à l'écran, s'afficher derrière tiles, doublé la taille en hauteur un sprite, doublé la taille en largeur, tester si le sprite est en collision avec un autre sprite ou tiles, le passer en mode multicouleur, et choix de la couleur personnelle du sprite.
Note : A cause de l'overlay, pour être visible, le sprite doit se trouver en px : 24 et py 50

* Copier les datas d'un sprites d'une adresse *

void set_sprite_data (**unsigned int** adr_cible, **unsigned char*** adr_data, **unsigned char** nb_sprite)	
unsigned int adr_cible	Adresse de destination.
unsigned char* adr_data	Adresse de lecture. (Tableau par exemple)
unsigned char nb_sprite	Nombre de sprite à copier.

Tous comme les tiles, la fonction suivante permet de copier un ou plusieurs sprite(s) contenu dans un tableau / une adresse / pointeur à une adresse que vous le souhaiter.

* Configurer le pointeur de pattern d'un sprite *

void set_pointers_sprite(**unsigned char** id_sprite,**unsigned char** value)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)
unsigned char value	Valeur du pointeur. (0-255)

Le pattern d'un sprite est encodé sur 64 octets. Le VIC à besoin de connaître l'emplacement de départ du pattern. Celui si doit se trouver dans la plage d'adresse accessible au VIC II.(Dans sa bank)
Le pointeur est défini pour chacun des 8 sprites. Il se situe dans les 8 derniers octets de la mémoire écran.
Pour configurer le pointeur de pattern au bonne endroit c'est simple. La formule c'est valeur du pointeur (0-255) * 64 + adresse de départ du VIC.
La fonction permet de simplifier la configuration du pointeur de sprite.

* Afficher un sprite à l'écran *

void show_sprite(**unsigned char** id_sprite);

void show_sprite(**unsigned char** id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Permet d'afficher le sprite voulu à l'écran.

==========
* Cacher un sprite à l'écran *
==========

void hide_sprite(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Permet de désactiver un sprite à l'écran.

==========
* Configurer la position d'un sprite à l'écran *
==========

void draw_sprite(unsigned char id_sprite, unsigned int position_x, unsigned char position_y)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)
unsigned int position_x	Position X du sprite
unsigned char position_y	Position Y du sprite

Permet de position d'un sprite à l'écran.
Attention un sprite à l'écran est visible à partir de 24 pixels horizontalement (X) et 50 pixels verticlament(Y).
La fonction gère tout seul le 9em bits de la position x pour dépasser les 255 pixel de l'écran. (C'est pour cela que la position X est un **unsigned INT** et par un **unsigned char**).

==========
* Doubler la hauteur d'un sprite *
==========

void double_height_sprite_on(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Le sprite est "doublé" sur la hauteur.

==========
* Désactiver le Doubler la hauteur d'un sprite *
==========

void double_height_sprite_off(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Le sprite n'est plus doublé sur la hauteur.

==========
* Doubler la largeur d'un sprite *
==========

void double_*width*_sprite_on(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Le sprite est doublé en largeur.

==========
* Désactiver le Doubler de largeur d'un sprite *
==========

void double_*width*_sprite_off(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Le mode "doubler" du sprite est désactivé.

==========
* Priorité d'un sprite par apport au tile *
==========

void sprite_priority_on(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Fait passer le sprite derrière tiles. Seul les pixel invisible du tiles affiche le sprite.

void sprite_priority_off(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Fait passer le sprite devant le tile. Seul les pixel invisible du sprite fait apparaître les couleurs du tile.

==========
* Détecter les collisions d'un sprite *
==========

Unsigned char get_collision_sprite()	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Collision Sprite/Sprite

Unsigned char get_collision_character()	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Collision sprite/tiles

une collision se produit quand le sprite en question se superpose à un pixel non transparent d'un sprite ou d'un tile (character). La fonction renvois 1 octet. Et en fonction si le bit de l'octet est 0 (pas de collision) ou 1 (collision) on peu connaître qu'elle sprite est en collision !

Bit activé	Sprite en collision
Bit 7	Sprite 7
Bit 6	Sprite 6
Bit 5	Sprite 5
Bit 4	Sprite 4
Bit 3	Sprite 3
Bit 2	Sprite 2
Bit 1	Sprite 1
Bit 0	Sprite 0

```
================================================
* Activer le Mode Multicolore pour les sprites *
================================================
```

void set_sprite_multicolore_on(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Tous comme les tiles, Chaque sprites peuvent passer en mode multicolore.

Dans ce mode il faut tous comme les tiles, 2 bits pour afficher un point. Ce qui permet 4 possibilités. (La couleur transparente, la couleur individuel du sprite, et deux couleurs choisis généralement pour les sprites.)

```
==================================================
* Désactiver le Mode Multicolore pour les sprites *
==================================================
```

void set_sprite_multicolore_off(unsigned char id_sprite)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)

Désactive le mode multicolore du sprite.

```
============================================
* Choix de la Couleur individuel du sprite *
============================================
```

void set_color_sprite(unsigned char id_sprite,unsigned char color_id)	
unsigned char id_sprite	Index du sprite à configurer (0 à 7)
unsigned char color_id	Couleur du sprite (0-15)

Permet de choisir la couleur du sprite.

```
====================================
* Choix de la Couleur 1 des sprites *
====================================
```

void set_sprite_color_1(unsigned char color_id)	
unsigned char color_id	Index de la couleur (0 à 15)

Permet de configurer la couleur 1 des sprites.

```
====================================
* Choix de la Couleur 2 des sprites *
====================================
```

void set_sprite_color_2(unsigned char color_id)	
unsigned char color_id	Index de la couleur (0 à 15)

Permet de configurer la couleur 2 des sprites.

* Les commandes *

* Tester les deux joystick *

Unsigned char get_joystick_1()
Renvois le résultat du joystick 1

Unsigned char get_joystick_2()
Renvois le résultat du joystick 2

Voici les defines pour savoir la position du joytisk.

DEFINE	REEL	DIRECTION
J_UP	1<<0	HAUT
J_DOWN	1<<1	BAS
J_LEFT	1<<2	GAUCHE
J_RIGHT	1<<3	DROITE
J_FIRE	1<<4	FEU

Exemple de code :

Code en C : Exemple Joystick 1

```
if (get_joystick_1() & J_FIRE)
{
  // Votre code si le bouton feu du joystick 1 est utilisé...
}
```

Attention :
Sur un vrais commodore 64 (et 128), ne pas brancher une manette de jeu megadrive/genesis car ça renvois du 5v et le cia n'aime pas ça du tout !

* Tester le clavier *

Unsigned char *get_keyboard_key()*
Renvois le résultat de la touche testé

cette fonction permet de récupérer une valeur de la touche du clavier utilisé.
Voir le tableau de macro à la fin du document pour utiliser les défines approprié au touche.

Numeros des touches en annexe

Code en C : Exemple Clavier

```
if (get_keyboard_key()==KEY_A)
{
  // Votre code si la touche A est utilisé.
}
```

==========
* Attendre une touche du clavier *
==========

void *wait_key(unsigned char id_key)*	
unsigned char id_key	Valeur de la touche testé.

Permet d'attendre l'appuie d'une touche avant de continuer le programme.
Note : l'utilisation de KEY_ANY permet d'attendre n'importe qu'elle touche.

==========
* Activer le touche shift *
==========

set_shift_on()
Active la touche shift

==========
* Désactiver la touche shift *
==========

set_shift_off()
Désactive la touche shift et évite le combo Commodore + shift pour changer les graphismes "minuscule/majuscule" en plein jeu donc bug graphique.

==========
* Tester les paddles *
==========

unsigned char get_paddle(unsigned char port,unsigned char id_paddle)	
unsigned char port	Port d'entrée (1 ou 2) du paddle sur le commodore 64. (Prise DB9)
unsigned char id_paddle	Numéro du paddle sur le port. (1 ou 2)
Renvois la position d'un paddle. (entre 0 et 255)	

==========
* Tester le bouton fire des paddles *
==========

unsigned char get_paddle_fire (unsigned char port,unsigned char id_paddle)	
unsigned char port	Port d'entrée (1 ou 2) du paddle sur le commodore 64. (Prise DB9)
unsigned char id_paddle	Numéro du paddle sur le port. (1 ou 2)

* La mémoire *

* Peek et Poke *

unsigned char PEEK(addr)	
addr	Adresse 16bits
Permet de lire une valeur 1 octet à l'adresse voulue	

unsigned int PEEKW(addr)	
addr	Adresse 16bits
Permet de lire une valeur sur 2 octets (un mot/word) à l'adresse voulue	

POKE(addr,val)	
addr	Adresse 16bits
val	Valeur sur 1 octet (0-255)
Permet écrire 1 octet à l'adresse voulue.	

POKEW(addr,val)	
addr	Adresse 16bits
val	Valeur sur 2 octet (0-65535)
Permet écrire 2 octets à l'adresse voulue.	

Note : Pour POKEW() et PEEKW(), la valeur 16 bits est décomposée dans deux octets à l'adresse 1 et adresse 2 Pour connaître la valeur 16 bit, la formule est octet de l' adresse 2 * 256 + contenue de l'adresse 1.

* Désactiver la rom basic *

Void set_loram_ram()
Désactive la rom Basic. Récupéré de la ram (8ko à partir de $A000)

Note : La Rom Basic est désactivé par défaut avec happyc64

* Active la rom basic *

Void set_loram_basic()
Active la Rom Basic.

Note : Normalement la rom du basic est désactivé nativement avec happyc64.

==========
* Désactiver la rom Kernal *
==========

Void set_hiram_ram()
Désactive la rom Kernal. Récupéré de la ram (8ko à partir de $E000

Attention à bien switcher correctement cette bank de donnée sous peine de bug ! (Je conseille au débutant de ne pas toucher à cette partie et de garder le Kernal activé. HappyC64 l'utilise pour la lecture des fichiers par exemple)

==========
* Activer la rom Kernal *
==========

Void set_hiram_kernal()
Désactive la rom Kernal. Récupéré de la ram (8ko à partir de $E000)

Note : Avant de désactiver le Kernal pour lire les valeurs qui se trouve à cette adresse, il faut désactiver les interruptions.
Il faut aussi réactiver les interruptions après avoir réactivé le kernal.

==========
* Lire un bit dans un octet *
==========

unsigned char get_bit(**unsigned char** id_bit,**unsigned char** value)	
id_bit	Position du bit dans l'octet. (0 à 7) 0: *1er bit à droite*
value	Valeur à tester comprise entre 0 et 255. (1 octet)
La fonction permet de tester un l'état d'un bit à une position voulue !	

==========
* Mettre à 1 le Nème bit *
==========

unsigned char set_bit(**unsigned char** id_bit,**unsigned char** value)	
id_bit	Position du bit dans l'octet. (0 à 7) 0 : *1er bit à droite*
value	Valeurs à modifier comprise entre 0 et 255. (1 octet)
La fonction permet de mettre un bit à 1 à la position voulue	

==========
* Mettre à 0 le Nème bit *
==========

unsigned char unset_bit(**unsigned char** id_bit,**unsigned char** value)	
id_bit	Position du bit dans l'octet. (0 à 7) 0 : *1er bit à droite*
value	Valeurs à modifier comprise entre 0 et 255. (1 octet)
La fonction permet de mettre un bit à 0 à la position voulue	

==========
* Le SID *
==========

Le Sid est le processeur sonore du C64. HappyC64 permet de sortir du son avec le Sid.

==========
* Modifier le volume sonore du C64 *
==========

void set_volume(unsigned char volume)	
unsigned char volume	Paramètre le volume general. (0-15)

==========
* Jouer un Son *
==========

```
void set_sound(
                unsigned char voice,//  VOICE_1,VOICE_2,VOICE_3
                unsigned char lb_freq,
                unsigned char hb_freq,
                unsigned char lb_pulse,
                unsigned char hb_pulse,
                unsigned char waveforme,// Utiliser TRIANGLE,SAWTOOTH...
                unsigned char attaque_decay,
                unsigned char sustain_release                    )
```

unsigned char voice	utiliser le macro VOICE_1 , VOICE_2 , VOICE_3
unsigned char lb_freq	
unsigned char hb_freq,	
unsigned char lb_pulse	
unsigned char hb_pulse	
unsigned char waveforme	Utiliser le macro TRIANGLE,SAWTOOTH...
unsigned char attaque_decay	
unsigned char sustain_release	

Define pour le numéro de canal	
Define	Valeur Réel
VOICE_1	0
VOICE_2	7
VOICE_3	14

Define pour la configuration du canal	
Define	Valeur Réel
TRIANGLE	17
SAWTOOTH	33
PULSE	65
NOISE	129

==========
* Divers *
==========

==========
* Générateur Pseudo Aléatoire 8 et 16 bits *
==========

unsigned char get_rnd(unsigned char nombre_max)	
unsigned char nombre_max	Valeur entre 0 et 255

unsigned int get_rnd16(unsigned int nombre_max)	
unsigned char nombre_max	Valeur entre 0 et 65535

Fonction qui retourne une valeur comprise entre 0 et nombre_max.
(Utilise le générateur de bruit du SID)

==========
* Décompression RLE *
==========

void rle_decrompression(unsigned int source,unsigned int destination)	
unsigned int source	Adresse des data à décompresser
unsigned int destination	Destination des data décompresser

HappyC64 possède une fonction simple pour décompresser des "data" au format RLE avec une contrainte ! Vos datas RLE doivent se finir par un octet qui vaut 0 pour mettre fin à la routine de décompression.

Le format RLE est simple : Voici une série d'octet au format RLE

2,4,1,0,7,5,4,4,0

Ce qui fait une fois décompressé !
4,4,0,5,5,5,5,5,4,4,4,4

==========
* Compression RLE *
==========

void rle_compression(unsigned int source,unsigned int destination, unsigned int size)	
unsigned int source	Adresse des data à décompresser
unsigned int destination	Destination des data décompresser
unsigned int size	Nombre d'octet dans le source à compresser.

La fonction : *void rle_compression(unsigned int source,unsigned int destination,unsigned int size)*
Permet d'appliquer une simple compression RLE. A la fin de la routine, 1 octet supplémentaire est ajouter avec la valeur 0.

* Cassette et Disquette *

* Sauvegarder des datas dans un fichier *

unsigned char save_file(**unsigned char*** name,**const void*** buffer, **unsigned int** size,**unsigned char** device)	
unsigned char* name	Le nom du fichier. un ,"option" permet de typer le fichier. d : del s : Sequenciel (seq) u : USR p : PRG (defaut) l : REL exemple : "data,s"
const void* buffer	Adresse source à sauvegarder
unsigned int size	Nombre d'octet dans le source à sauvegarder (2 octets est ajouté sur la disquette/cassette)
unsigned char device	Id du device. 1 pour cassette, 8 pour lecteur disquette 1, 9 pour lecteur disquette 2 ...

Permet de sauvegarder des datas binaires dans un fichier sur disquette (ou cassette).
2 octets sont ajoutés au début du fichier.

* Charger des datas dans un fichier *

unsigned int *load_file(const char*name, const void* buffer, unsigned char device)*	
unsigned char* name	Le nom du fichier. un ,"option" permet de typer le fichier. d : del s : Sequenciel (seq) u : USR p : PRG (defaut) l : REL exemple : "data,s"
const void* buffer	Adresse de destination des datas
unsigned char device	Id du device. 1 pour cassette, 8 pour lecteur disquette 1, 9 pour lecteur disquette 2 ...

Cette fonction permet de charger les données binaire d'un fichier dans un tableau (ou à partir d'une adresse mémoire)

Note : Les deux octets du "header" ne sont pas enregistré dans le buffer.

Note 2 :La Fonction renvois le nombre octet chargé. Si le nombre est égale à 0, alors il y a une erreur de chargement.
Erreur à récupéré avec get_error()

Code en C : Exemple Ouverture de fichier

```
if (load_file("map1,s",(void*)0x8000,8)!=0)
{
  error = get_error() ;
}
```

==========
* Gestion des erreurs *
==========

***unsigned** get_error()*

Renvois un code erreur avec load_files()

==========
* Un petit mode d'emplois sur les pointeurs *
==========

Les deux fonctions pour manipuler les datas dans un fichier utilise une adresse mémoire de départ pour la lecture, ou la sauvegarde. C'est utile pour charger/sauvegarder des datas à un emplacement voulu, le buffer demande un pointeur. Voici comment les gérer si vous n'avez pas l'habitude de cela.

Le nom d'un tableau est pointeur. Donc un tableau avec le nom buffer[] : save_file("sauvegarde,s",buffer,128,8) ; sauvegardera dans le fichier sauvegarde.seq, 128 octets du tableau buffer sur la disquette…

On peux utiliser un pointeur avec une adresse définie (ou autre…)
unsigned char **buffer =(char)**0xC000

Dans ce cas la, en utilisant le mot buffer on sauvegarde 128 octets à partir de l'adresse 0XC000.

On peux placer une adresse directe dans la fonction aussi avec un cast !

Voici l'exemple avec 0xC000
save_file("sauvegarde,s",(void*)0xC000,128,8);

Les Interruptions

Configurer la fonction irq

void init_adr_irq(unsigned int adresse);
Permet de choisir une fonction qui sera appelé à chaque interruption

Permet de choisir la fonction voulue qui sera appelé à chaque interruption.
Exemple : *init_adr_irq((int)*compteur) ;*

Instruction de fin de fonction

end_fonction_irq()
A placer à la fin de votre fonction qui sera appelé par les interruptions

Code Exemple

Code C : Exemple interruption simple

```
unsigned char cp ;

void compteur()
{
  // A chaque interruption, cp est incrémenté de 1
  cp++ ;
  end_fonction_irq() ;
}

main()
{
  init_adr_irq((int)*compteur);

  while(1)
  {

  }


}
```

* Text Engine *

Le SDK permet d'afficher du texte. Ceci dit une petite préparation est à effectué. La représentation du texte dans la mémoire de character doit avoir une certaine organisation.
- Il suit l'organisation ASCII
- Le premier élément c'est "l'espace".
- Vous n'êtes pas obligé de placer les majuscules et minuscules pour gagner de la place. Vous pouvez représenter des minuscules dans la partie majuscule mais vous devez écrire vos textes en majuscule !
- Vous devez garder le pattern du mode vidéo appliqué. Un A en mode standard n'aura pas la même gueule qu'un A en mode Multicolor !

* Ordre ASCII HappyC64 *

Voici l'ordre de vos fonts pour happyC64 et les numéro ASCII,
(Sp = Espace)
Il faut 93 tiles pour mémoriser la table complète.

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	Égal	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125		
p	q	r	s	t	u	v	y	z	y	z	{	|	}		

* Modifier le pointeur de texte *

void set_text_pointer(**unsigned char** pointeur)	
unsigned char *pointeur*	Index du pattern pour l'espace (0-255)

Cette fonction vous permet de redéfinir la place du premier caractère (l'espace) dans votre plage mémoire de pointeur de caractère. Exemple à 0, votre jeu de police d'écriture est au début de la plage des caractères. A 5, le départ sera le 6em caractères.

==========
* Afficher un texte à l'écran *
==========

void draw_text (unsigned char px,unsigned char py, unsigned char* text, unsigned char color , unsigned char slow_wait_letter)	
unsigned char px	Position X du texte. (Case) (0-255)
unsigned char py	Position Y du texte. (Case) (0-255)
unsigned char* text	Pointeur / Chaîne du caractères. (Mode ASCII)
unsigned char color	Index de couleur du texte. (0-15)
unsigned char slow_wait_letter	Permet d'attendre X waitvbl() entre l'affichage d'une lettre. (Permet d'afficher du texte lettre par lettre)

Cette fonction permet d'afficher du texte à l'écran.

Exemple : draw_text(0,0,"Hello World",5,0);
Un Hello World jaune s'affiche en 0,0

==========
* Afficher un bloc de text à l'écran *
==========

void draw_bloc_text (unsigned char px, unsigned char py, unsigned char* text, unsigned char color, unsigned char size_ligne, unsigned char slow_wait_letter)	
unsigned char px	Position X du texte. (Case) (0-255)
unsigned char py	Positin Y du texte. (Case) (0-255)
unsigned char* text	Pointeur / Chaîne du caractères. (Mode ASCII)
unsigned char color	Index de couleur du texte. (0-15)
unsigned char size_ligne	Taille d'une ligne.
unsigned char slow_wait_letter	Permet d'attendre X waitvbl() entre l'affichage d'une lettre. (Permet d'afficher du texte lettre par lettre)

Note sur le size_ligne : Permet de paramétrer un saut ligne automatique. C'est aussi utile quand vous utilisez la fonction @C pour effacer le bloc de texte.

Draw_bloc_text permet d'utiliser un code de fonction :

Code	Fonction
@S	*Force un saut de ligne.*
@W	*Demande un appuie de n'importe qu'elle touche pour la suite du texte.*
@C	*Permet d'effacer le texte. L'effacement se fait en fonction du size_ligne en largeur et du derniers position Y du texte. Notons que la prochaine lettre se place au coordonné de départ. Le prochaine lettre à afficher et bien la lettre suivant @C*

Note : Vous ne pouvez pas utiliser la lettre @ dans votre texte ! Risque de bug.

```
=====================================================
* Afficher une valeur à l'écran 8 bits et 16 bits *
=====================================================
```

void draw_text_value_8 (**unsigned char** px,**unsigned char** py,**unsigned char** valeur,**unsigned char** color)	
unsigned char px	Position X du nombre. (Case) (0-255)
unsigned char py	Position Y du nombre. (Case) (0-255)
unsigned char valeur	Permet d'afficher une valeur sur (8bits)
unsigned char color	Index de couleur du nombre(0-15)

void draw_text_value_16 (**unsigned char** px,**unsigned char** py,**unsigned int** valeur,**unsigned char** color)	
unsigned char px	Position X du nombre. (Case) (0-255)
unsigned char py	Position Y du nombre. (Case) (0-255)
unsigned int valeur	Permet d'afficher une valeur sur (16 bits)
unsigned char color	Index de couleur du nombre(0-15)

* REU : RAM EXPANSSION UNITE *

* Introduction *

Les extensions REU sont des modules mémoire supplémentaires pour le commodore 64. Trois modules officiel existe. Le 1700 (128KO),1750(512KO), et 1764(256ko). (L'émulateur VICE permet de simuler un module extension. Pour information, C64 forever (qui utilise vice) permet de choisis entre un 256ko et 512ko.

Le module fonctionne par DMA. c.a.d il va copier la partie voulue de la Ram du C64 pour la placer dans l'extension et inversement. (Contrairement au CPC6128 qui fonctionne en mode bank switching pour sa ram supplémentaire).

On ne peux adresse que 64ko dans le REU, pour adresse les 128 à 512ko, il faut aussi designer une "page" du REU dans les commandes de configurations. Par exemple, si je choisis l'adresse $400 de la page 0, on va taper à l'adresse physique $400 du REU, mais si on choisis l'adresse $400 de la page 1, on tapera à l'adresse physique du reu en $10400.

Dans les registres du REU, il y a 1 octets qui permet de choisir la page soit au maximum 256 pages. (On peux donc utiliser des extensions REU qui vont jusque 16 Mo. La cartouche ultimate 2+ doit exploiter ça.

* Configurer l'adresse du C64 *

void reu_set_adresse_c64(**unsigned int** adresse)	
unsigned int adresse	Adresse dans le C64 ($0 à $FFFF)

Cette fonction permet de configurer l'adresse de départ de travaille sur le commodore C64.

Note pour l'écriture d'une adresse : Ce n'est pas un "pointeur". Utiliser directement une valeur. Par exemple 0x400 ou une variable int qui contient la valeur 0x400.

* Configurer l'adresse du REU *

void reu_set_adresse_reu(**unsigned int** adresse,**unsigned char** id_bank)	
unsigned int adresse	Adresse offset dans le reu ($0 à $FFFF)
unsigned char id_bank	Numeros de la page dans le reu (0 à 255)

Cette fonction permet de configurer l'adresse de départ de travaille du reu. Elle comprend deux valeurs, une adresse entre $0 et $FFFF (qui est un offset) et la page du reu entre 0 et 255. (Attention, une extension 256ko comprend moins de page qu'une extension de 512ko). Une page c'est 64ko. Une extension de 256ko comprend donc 4 pages en tous. (Donc entre 0 et 3)

Note pour l'écriture d'une adresse : Ce n'est pas un "pointeur". Utiliser directement une valeur. Par exemple 0x400 ou une variable int qui contient la valeur 0x400.

* Plage d'octet à travailler *

void reu_set_size(unsigned int size)	
unsigned int size	Nombre d'octet à travailler.

Nombre d'octet que l'extension va travailler au moment du transfère. Si il faut transférer que 1ko utilisé la valeur 1024 par exemple.

* Lancer le transfère *

void reu_start_dma(unsigned char value)	
unsigned char value	Valeur qui représente le type de transfère

Cette commande permet de lancer le transfère de Ram. 3 defines à placer dans **value** est préparé.

Define pour le reu_start_dma		
#Define	Valeur Réel	Effet
MODE_C64_REU	0b10010000	Lance le transfère du C64 => REU
MODE_REU_C64	0b10010001	Lance le transfère du REU => C64
MODE_SWAP	0b10010010	Lance le transfère en mode échange. *REU=>C64 et en même temps C64=>REu*

Note sur le SWAP : Les datas sont échangés entre le C64 et le REU. La Cible prend les donnés de la source, et la source prend les donnés de la cible...

Pour les deux autres commandes, ce qui se trouve dans la cible est perdu puis qu'il est remplacé par les nouvelles donnés mais les donnés de la source ne sont pas effacé.

* Index des touches du clavier *

Touche Alphabétique		
#DEFINE	**RÉEL**	**TOUCHE**
KEY_A	10	A
KEY_B	28	B
KEY_C	20	C
KEY_D	18	D
KEY_E	14	E
KEY_F	21	F
KEY_G	26	G
KEY_H	29	H
KEY_I	33	I
KEY_J	34	J
KEY_K	37	K
KEY_L	42	L
KEY_M	36	M
KEY_N	39	N
KEY_O	38	O
KEY_P	41	P
KEY_Q	62	Q
KEY_R	17	R
KEY_S	13	S
KEY_T	22	T
KEY_U	30	U
KEY_V	31	V
KEY_W	9	W
KEY_X	23	X
KEY_Y	25	Y
KEY_Z	12	Z

Valeurs Numériques		
#DEFINE	**Réel**	**Touche**
KEY_0	35	0
KEY_1	56	1
KEY_2	59	2
KEY_3	8	3
KEY_4	11	4
KEY_5	16	5
KEY_6	19	6
KEY_7	24	7
KEY_8	27	8
KEY_9	32	9

Touche Divers		
Define	Réel	Touche
KEY_L_ARR	57	<-
KEY_CLR	51	
KEY_DEL	0	Del
KEY_RET	1	Retourn
KEY_DN	4	
KEY_RT	2	
KEY_STOP	63	
KEY_SPC	60	Espace
KEY_EMPTY	64	Pas de touche

Touche Arithmétiques		
Define	Réel	Touche
KEY_PLUS	40	+
KEY_MOINS	43	-
KEY_DIVISER	48	/
KEY_MULTIPLIER	49	*

Touche Fonctions		
Define	Réel	Touche
KEY_F1	4	F1
KEY_F3	5	F2
KEY_F5	6	F3
KEY_F7	3	F4

* La carte mémoire utile du C64 pour HappyC64 *

Le **Commodore 64** possède **64ko** (*65536 octets*) de mémoire ram. (Adresse $0 à $FFFF). Le C64 est découpé en 4 plages de **16ko** (*4096 octets*). (Bank).

Cette documentation va tenter de vous apporter des informations utiles pour HappyC64 et la gestion de sa mémoire.

* Des variables exploitables avant le memory screen de base *

Il existe des emplacements libres et exploitable dans la première partie de la ram. Des octets par ici, des octets par la ce qui peut être très pratique pour grappiller des "variable manuel".

Mémoire disponible dans la partie System	
$0042 ,$0052,$033B	Trois emplacements d'**1 octet** de disponible.
$00FB à $00FE	Une plage de **89 octets** disponible
$033C à $03FB	Une plage de **192 octets** disponible. *Note ceci est le **buffer datasette**. Si vous re charger des datas par cassette la plage sera effacer.*
$0CFC à $03FF	Une plage de **4 octets** disponible

* Le Memory Screen en configuration de base *

A l'allumage du commodore 64, le **memory screen** se trouve à l'adresse **$400** et prend fin à l'adresse l'adresse **$7FF**. Une plage **1000 octets** est réservé pour l'affichage vidéo. Les **8** derniers octets de cette plage sont réservés pour les pointeurs de sprite.

Screen Memory		
Adresse de départ	Offset	Description
$400	0	Début de la mémoire écran
$7E7	999	Fin de la mémoire écran
$7E8 à $7F7	1000	Zone libre de 15 octets
$7F8	1016	Pointeur de sprite 0
$7F9	1017	Pointeur de sprite 1
$7FA	1018	Pointeur de sprite 2
$7FB	1019	Pointeur de sprite 3
$7FC	1020	Pointeur de sprite 4
$7FD	1021	Pointeur de sprite 5
$7FE	1022	Pointeur de sprite 6
$7FF	1023	Pointeur de sprite 7

En déplaçant le **VIC2** ou / et le **memory screen**, il est possible de récupérer cette espace mémoire pour votre programme.

==========
*** Plage pour votre programme ***
==========

C'est à partir de l'adresse **$800** que votre programme est mémorisé avec happyc64. (Cassette ou Disquette) *L'adresse est choisis dans le fichier .cfg de CC65 pour information.*
Il y a 2 octets pour le header de lancement. Et le programme en lui même débute en **$802**.
Votre programme comprend le HappyC64, (il faut bien mémoriser la librairie) et votre code.

La limite de vos programmes à ne pas franchir est **$D000** et en sachant que CC65 (et le fichier de configuration de base que nous utilisons) place une pile de **2ko** avant **D000** ce qui revient à ne pas utiliser les adresses au dessus de **C800**.

==========
*** Plage du VIC en Standard ***
==========

Le Vic est configuré en standard à l'adresse **$0** de la mémoire du C64. Le vic est capable d'adressé **16ko** de mémoire Ram (et l'espace mémoire de la color map).
Configuré de base, le vic peut donc lire que la plage d'adresse comprise entre **$0** et **$3FFF** !
Ce qui veux dire que les patterns de sprite + patterns de tiles doivent se trouver à la fin de la taille de votre programme sans dépasser $3FFF dans une configuration standard qui est vivement conseillé de modifier).

==========
*** Plage de 16ko de libre ***
==========

16 ko de ram sont disponibles entre l'adresse **$4000** et **$7FFF**.
(Le Vic II peut être déplacé dans ce bloc !)

==========
*** Deux bloc de 8ko sont disponibles ***
==========

Les deux plages d'adresses comprises entre **$8000** à **$9FFF** et **$A000** à **$BFFF**.
16ko découpés en 2 Bloc de **8ko**.

Le premier bloc est prévus pour une cartouche de **8ko**. Mais reste disponible pour de la ram.

Le 2nd bloc est prévus pour 8ko aussi de cartouche (cartouche de 16ko) et la rom basic.
Ceci dit la rom basic est désactivé ce qui permet de récupéré **8ko** supplémentaires de ram.
(Le vic II peut être déplacé dans ce bloc qui peut être une place de choix pour nos programme avec happyC64 et en plaçant le memory screen au début du bloc ou pourquoi pas à la fin).

==========
*** Bloc de 4 ko de libre ***
==========

4 ko est libre de l'adresse $**C000** à **$CFFF**.
Ceci dit CC65 place **2ko** pour la pile à la fin de ce bloc.
Il ne faut donc pas utiliser la mémoire à partir de **C800**.
Le vicII peut être déplacé en **$C000** si vous voulez utiliser les 8ko du kernal pour la mémoire video.

* IO/Chara et Kernal *

Les deux dernières bank de la ram sont partagés par **8ko** dédiés au i/o, et à la rom de charactère :(**$D000**) et le Kernal (**$E000**)
Je suis bien tenté de dire pas touche à ça petit con.
Il est possible de désactivé le Kernal pour retrouver **8ko de Ram** mais ceci dit, ça plante si vous ne faite pas gaffe. (Il y en a qui ont essayé, et ils ont eu des problèmes)

==========
* ASTUCES *
==========

==========
* Le VIC-II dans la bank du Kernal *
==========

Le **kernal** débute à l'adresse **$E000** jusque à la fin de la ram du c64 $FFFF.
Le Kernal c'est le system exploitation du commodore 64, c'est une ROM. c.a.d que les informations sont seulement en lecture en **$E000**, il y a aussi de la RAM ou nous pouvons lire et écrire qui se trouve à notre disposition. Un truc cool c'est que si nous voulons écrire dans un emplacement d'une ROM, ba ça passe directement en RAM. Mais l'inverse non. Si la rom est activé, si on veux lire, c'est dans la rom que nous allons taper.

Il est possible de désactiver la ROM du Kernal pour lire la RAM. Ceci dit, sans faire attention, ça fait buger le programme. Je conseille donc de ne pas faire ça. Ceci dit c'est **8ko** de ram perdu ! Ou pas. Il est possible d'utiliser les 8ko de ram en tant que mémoire video et même sans désactiver le kernal. Le vic-II lui ira chercher les donnés en RAM directement, et comme on peux écrire directement en RAM sans désactiver la rom.

- Déplacer le vic-2 dans la dernier bank mémoire du vic. -

Le vic 2 ne peux adresser que **16ko** de mémoire. A la mise sous tension de la machine, il se trouve au début de la ram. **$0000**
Il faut le déplacer à la fin de la ram dans l'une des 4 positions possibles du vic-2 à l'adresse **$C000**

La commande à utiliser est : *set_vic_bank(VIC_BANK_3);*

- Déplacer le screen memory au début de la plage du kernal -

Pour le moment le screen memory se trouve en offset **$400** par apport à l'adresse du vicII soit à l'adresse **$C400**

Cela ne va pas, il faut le déplacer dans la partie occupé du kernal en $E000
Une fonction permet de déplacer le screen memory, on va donc le déplacer à l'offset **$2000** par apport au VIC_II (**$C000 + $2000 = $E000)**

set_adresse_screen_memory(SM_2000) ;

Il est possible de le déplacer à la fin avec un un **SM_3C00**. (**$FC00**)

- Déplacer le pointeur de character -

C'est la 3em manipulation à faire. Déplacer le pointeur de charactère pour lire les tiles dans la bank du kernal.

Il y a quatre possibilités pour les character.
La fonction pour déplacer le pointeur est set_location_character(id)
les 4 possibilité sont :

8 : pour placer les tiles en $E000 donc en début kernal
10 : pour les placer en $E800. (Il y a une plage de 400 octet de libre en le screen memory et le premier tiles)
12 : pour placer les tiles en $F000
14 : pour placer les tiles en $F800

- Mots sur le pointeur de sprite !!! -

Les pointeur de sprite (0) débute au début du VIC2 en **$C000.**
Vous avez 2ko de libre pour placer des pattern à cette endroit avant d'être dans la zone de la pile pour votre programme C. (32 patterns).

Vous pouvez placer vos pattern de sprite dans le Kernal. Mais attention à ne pas prendre l'adresse **$E000** comme offset 0 pour calculer votre emplacement des sprites et de bien configurer la machine pour faire cohabiter pattern de tiles et de sprites. (Un pattern de sprite c'est 64 octets).

- Mots sur le pointeur de tiles !!! -

Tous comme les sprites, il est possible de placer les tiles $C000 (Cela fait 256 patterns de tiles).

Ceci dit la zone du kernal fait 8ko, on peux très bien regrouper le screen memory, les patterns de tiles et les patterns de sprites en même temps pour libérer un max de place pour votre programme.

- Notre nouvelle ram card -

Notre nouvelle ram card se décompose donc comme ceci :

ADRESSE	TAILLE	NOTES
$0000 à $03FF	1ko	Variable System
$0400 à $07FF	1ko	Ram libre
$0800 à $3FFF	14ko	pour votre programme ou autre data.(**Début du programme**)
$4000 à $7FFF	16ko	pour votre programme ou autre data.
$8000 à $BFFF	16ko	pour votre programme ou autre data.
$C000 à $C7FF	2ko	pour votre programme ou autre data.
$C800 à $CFFF	2ko	PILE pour votre programme en C. (Variable Local)
$D000 à $DFFF	4ko	Variable I/O (Pas touche à ça petit con!!!
$E000 à $FFFF	8ko	KERNAL et MÉMOIRE VIDÉO (SCREEN MEMORY et PATTERN)

Vous avez un espace de 43 ko pour créer votre jeu en bien vous organisant. Notons que votre programme se loge à l'adresse **$800** ce qui vous laisse une bonne **40en de ko** pour écrire votre programme si vous désirez ne pas passer par un système de chargement de fichier data en plein jeu.

* Binaire / Décimale / Hexadécimale *

* Valeur Décimale *

Nous avons l'habitude de ce système de notation. Nous l'utilisons à longueur de journée. Le système décimale travaille en base de 10. C'est à dire qu'une unité à le choix de 10 valeurs. La valeurs souhaiter est comprise entre 0 et 9. (0,1,2,3,4,5,6,7,8,9) ce qui fait bien 10 possibilités.

Quand on ajoute 1 à 9, notre unité repart à 0, et l'unité de gauche, une valeur de 1 est ajouté à l'unité de gauche.

Exemple : *9 + 1 = 10*
ou pour mieux comprendre : 09 + 1 = 10
autre exemple : 99 + 1 = 100
décomposition : 099 + 1 =0(9+1)0
=100

Sur happyC64 l'écriture décimale se fait simplement en écrivant nos valeurs directement sans pré fixe.

* Valeur Binaire *

Les valeurs binaires sont la base de l'informatique. Pour comprendre le principe, c'est très simple, soit le courant passe, ou le courant ne passe pas. On peux voir ça aussi en fonction de la lumière. Il y a de la lumière ou pas.
…
Le système binaire travaille sur une base de 2. Une unité à que deux possibilités. Une valeur comprise entre 0 et 1.
Tous comme les valeurs décimale. Ajout 1 à la valeur 1 revient à 0 dans notre unité et à ajouté 1 à l'unité de gauche.

Exemple : 1+0 =   1
: 1+1 = 10
: 11+1 = 100

Sur HappyC64 (et surtout CC65) nous pouvons utiliser directement des valeurs binaires. Pour cela il faut utiliser le préfixe **0b**

Exemple : 0b10100101

Dans les assembleurs ou représentation, une valeur binaire peut avoir comme préfixe **%**

Exemple :%10100101

* Valeur Hexadécimale *

3em forme très utilisé en programmation. La forme Hexadécimale qui est une base de 16. Donc une unité permet 16 valeurs. Pour son écriture, après le 9, nous utilisons les lettres de l'alphabet. Ce qui fait :
(0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

Ajouter 1 à 9, ne donne pas 10 mais A.
Ajouter 1 à F donne 10. On passe l'unité à 0 et on ajoute 1 à l'unité de gauche.

Sur HappyC64 nous pouvons aussi utiliser directement des valeurs haxadécimale.
Pour cela il faut utiliser le préfixe **0x**
exemple : 0x400

Notons que dans d'autre langage ou "norme", vous pouvez rencontrer le préfixe **$**
$400

voir un **H** au début du nombre. H400

Les valeurs décimale sont très très utilisé pour les adresses dans la ram.
Sur Commodore 64, l'adresse débute à 0x0 pour finir en 0xFFFF

==========
*** L'astuce des nombres ***
==========

Voici des petites tipes à comprendre.

1 octet c'est une valeurs comprise entre 0 et 255 en décimale.
C'est aussi une valeur comprise entre 0 et 11111111 en valeurs binaire. (8 bits)
et aussi une valeurs comprise entre 0 et FF en Hexadécimale.

Si on décompose FF en hexadécimale,C'est bien 4 bits à gauche et 4 bits à droite.

Hexadécimale : F/F
Binaire :1111/1111
Décimale :15 / 15
(255)

Si on décompose F1
Hexadécimale : F/1
Binaire :1111/0001
Décimale :15 / 1
(241)

==========
*** Décalage de bit ! ***
==========
Très utile en programmation. Sachez que votre commodore 64 ne sait pas faire de multiplication et de division. Demander à votre C64 de faire ce genre d'opération revient en gros à faire des additions à gogo. Donc on mange du temps machine et de la ram.

Par contre votre commodore 64 connaît une technique simple qui est de décaler des bits d'un octet à droite ou à gauche.
Et ça, il fait très très très bien le faire et rapidement surtout.

A vrais dire je dis, le commodore 64, mais c'est le processeur de l'ordi qui fait ça…

Nous allons voir deux / trois trucs avec le décalage de bit.

Etude de cas :
%00000001
Cette valeur binaire représente 1 décimale.
Maintenant nous allons décaler le "1" binaire d'une unité à gauche.

%00000010
ce qui représente %10 soit 2

Nous allons continuer de nouveau en décalant encore d'une unité à gauche.
%00000100
soit%100 qui est égale à 4.
Ce qui nous donne la suite suivant. 1,2,4 puis 8,16,32,64,128,256
Nous pouvons voir que décaler les bits binaires d'une unité revient à multiplier par 2. De déclarer de deux unités, revient à multiplier par 4…

En C le signe de décalage vers la gauche est **<<**
Et le décalage à droite est **>>**

1<<1 revient à faire 1 x 2.
16<<2 revient à faire 1x4
32>>4 revient à faire 32/4

Notons que le décalage de bit n'est pas une opération prioritaire.
1>>2x2 donnera 1>>4
Penser à mettre des parenthèses dans les décalages de bit.
(16>>2)x2 = (16/4)x2 = 4x2 = 8

Voilà la raison que dans le monde rétro, on aime travaille en base de 2 pour facilité le décalage de bit et le travaille sur les nombres.

* Commandes basic utiles *

* Gestion des disquettes *

Les programmes de HappyC64 sont transformés directement en ASM. La lancement des disquettes sur émulateur sont souvent automatisées. Sur un vrais C64, ce n'est pas le cas. Voici un petit guide rapide de gestion de disquettes.

Chaque périphérique porte un numéro d'identification. Pour le lecteur de disquette, le numéro id débute à 8 pour le lecteur 1 jusque à 30.

La commande de chargement de donnée en mémoire :
load *"nom du fichier"*,**numero du peripherique,***adresse*
Ce qui donne par exemple :
load "prog",8,1

Voici des exemples :

Par défaut j'utilise le périphérique 8

```
------------------------------------------
* Charger le premier programme en mémoire *
------------------------------------------
load "*",8,1

---------------------------------------
* Charger le fichier "prog" en mémoire *
---------------------------------------
load "prog",8,1

-----------------------
* Lancer le programme *
-----------------------
run

-------------------------------------------------
* Charger le contenu de la disquette en mémoire *
-------------------------------------------------
load"$",8

-----------------------------------------------------------
* Lister le contenue de la disquette après chargement *
-----------------------------------------------------------
list

--------------------------
* Formater une disquette *
--------------------------
OPEN 15,8,15:PRINT#15,"NEW:NOM,2A":CLOSE 15
```