Le code 65816 de la Rom v 0.01 à nu

Ceci est un pseudo-fichier source du code 65816 v 0.01 (extrait de la Rom Super-Oric "Base1.bin"). Pour obtenir un fichier source opérationnel, il faudrait retirer les adresses et les codes hexadécimaux (les deux premières colonnes) et le plus gros des commentaires. Attention, lorsqu'un commentaire court sur plusieurs lignes, seule la première ligne comporte un point-virgule (indiquant une remarque). Si vous voulez conserver ces commentaires longs, il faudra ajouter des points-virgules au début de chaque ligne du vrai fichier source.

Ce pseudo-fichier source a été élaboré en s'inspirant du fichier source Oric2.s de la Rom Super-Oric version 1.46, de Fabrice F. et en y reportant les différences, dans la partie code 65816, entre la Rom Super-Oric version 1.46 et la Rom Super-Oric Base1.bin (ou version 0.01). Quelques modifications sont dues à des déplacements de routines, donc de la valeur des labels. Il n'y a donc rien à faire, l'assembleur s'en chargera.

En fait la cartouche Base1.bin commence par le "code 65816" (#231 octets), puis les #4000 octets de la Rom Basic, puis des #00 et enfin la zone "copyright". Ce code 65816 sert principalement à configurer la Snes (choix du type d'écrans Text et Hires, gestion du joypad, rafraîchissement de la mémoire vidéo Vram, copie de la Rom Basic dans le haut de la banque principale $7E, etc., etc.) pour rendre possible le fonctionnement de la Rom Basic...

ATTENTION! Pour compiler la Rom Super-Oric version 0.01, il faut désactiver la ligne fixcart.txt dans buildrom.bat car elle est inadaptée à la vieille structure de la Rom (code 65816 puis Basic). Editer plutôt manuellement le fichier ORIC2.SMC, qui est produit: Il faut récupérer les #8000 octets de la Rom Super-Oric proprement dite et modifier correctement le copyright.

Le fichier proprement dit

```
                        NAM Super-Oric by F.F   ; sera placé en $7FC0 de la Rom Super-Oric
                        ORG $008000             ; adresse initiale de la Rom Super-Oric dans la banque $00
                        COU 2                   ; ?

008000 5C 08 80 80      JMP.L start+$800000     ; Exécution plus rapide en banque $80
; Cette routine est exécutée au boot. Les #8000 octets de la Rom Super-Oric sont situés de $008000 à $00FFFF (adresse
initiale dans la mémoire Flash, c'est à dire en banque $00 de la Snes. Les #4000 octets de la partie "Basic" sont copiés
de $7EC000 à $7EFFFF (adresse d'exécution en Ram). Le "code 65816" situé au départ de $008000 à $008230 (dans la
mémoire Flash) est exécuté en banque $80 de $808000 à $808230 (adresse d'exécution accès rapide).

008004  20 68 81        JSR hires_transfer      ; en $8168
; Cette routine est seulement appelée par REFRESH à partir de la Rom Basic en banque $7E. Elle lance le transfert
DMA de l'écran Hires (situé en banque $7F) vers la Ram vidéo (Vram).
008007  6B              RTL             ; Retour à l'appelant en banque $7E

; ********************************************************

        start                           ; Effectue un certain nombre d'initialisations au lancement
; Cette routine est exécutée au boot.
008008  18              CLC             ; Pour initialiser le mode d'émulation
008009 FB               XCE             ; Permute les bits Carry et Emulation (0 = Mode Native)
```

```
00800A  C2 10        REP #$10              ; X,Y sur 16 bits
00800C  E2 20        SEP #$20              ; A/M sur 8 bits
00800E  20 AA 81     JSR init_ppu          ; en $81AA
008011  20 A8 80     JSR clear_vram        ; en $80A8
008014  20 55 80     JSR init_charset      ; en $8055
008017  20 36 80     JSR init_graphscreen  ; en $8036
00801A  20 BC 80     JSR init_palette      ; en $80BC
00801D  A9 0F        LDA #$0F              ; Pour allumer l'écran avec la plus forte luminosité
00801F  8D 00 21     STA $2100             ; INIDISP (INIt DISplay) Screen Display Register
008022  A9 7E        LDA #$7E              ; Pour banque $7E (banque Oric principale)
008024  48           PHA                   ; et empile pour initialiser DBR
008025  AB           PLB                   ; dépile dans le Registre de Banque de Données (DBR)
; Important : Le DBR (Registre de Banque de Données) est initialisé sur la banque $7E (principale).
008026  20 61 80     JSR copyrom           ; en $8061
008029  E2 30        SEP #$30              ; A/M X/Y sur 8 bits
00802B  A9 01        LDA #$01              ; A=#$01
00802D  EB           XBA                   ; B=#$01 afin que les TCS laissent S en page 1
; XBA = Exchange B and A 8-bit Accumulators
00802E  A9 00        LDA #$00              ;
008030  AA           TAX                   ; X=#$00
008031  A8           TAY                   ; Y=#$00
008032  5C 8F F8 7E  JMP $7EF88F           ; cold reset Basic Oric situé dans la banque principale $7E


; ********************************************************

        init_graphscreen                   ; initialise l'écran Hires
; Cette routine n'est appelée que par la routine "start" lors du boot, avec X,Y sur 16 bits et A sur 8 bits.
; L'écran Hires a une taille de 256x224 pixels soit 57344 pixels (#E000). Ceci représente 32x28 "tuiles" de 8x8 pixels soit 896 "tuiles" (#380) Une "tuile" compte 8x8=64 (#40) pixels Je ne sais pas si ces données aident à comprendre la routine obscure qui suit et que pour ma part je ne comprends pas.
008036  A2 00 7C     LDX.W #$7C00          ; initialisation BG1 (X=$7C00 est une adresse dans la Vram)
008039  8E 16 21     STX $2116             ; mis en $2116 et en $2117 (2 octets)
; $2116 VMADDL et $2117 VMADDH = adresse pour la lecture et écriture en Vram. A chaque fois que la donnée est écrite ou lue dans la Vram, l'adresse sera incrémentée automatiquement. La valeur d'incrémentation est déterminée par le registre $2115, qui n'ayant pas été mis à jour dans cette routine, date de l'une des routines précédemment appelées, par exemple dma_screen ou hires_transfer ou init_ppu (situation malsaine). Si j'ai bien compris, cette adresse initiale $7C00 dans la Vram, correspond au mode BG1. Les données mises ensuite en $2118 sont alors écrites dans la zone BG1 de la Vram, à partir de $7C00 et l'adresse est alors incrémentée pour la prochaine écriture. Sauf que selon le code de la routine hires_transfer (en $008168), la Vram est accédée en écriture de $0800 à $E7FF pour l'écran Hires (#$E000 octets). La présente écriture "initialisation BG1" tombe au milieu de cet écran Hires… à revoir !
00803C  A2 40 00     LDX.W #$0040          ; X=#0040 sur deux octets = décimal 64
        loop1_bg1
00803F  8E 18 21     STX $2118             ; mis en $2118 et en $2119 (2 octets)
; $2118 VMDATAL et $2119 VMDATAH = données pour l'écriture dans la VRAM.
; Je ne comprends pas pourquoi cette valeur variable de #0040 à #$03FF est écrite dans la Vram.
008042  E8           INX                   ; X va donc de #0040 à #03FF
008043  E0 00 04     CPX.W #$0400          ; = décimal 1024 (ni le pourquoi de ce #0400)
008046  D0 F7        BNE loop1_bg1         ; BNE $803F reboucle tant que #0400 n'est pas atteint
008048  A2 00 00     LDX.W #$0000          ; X=#0000 sur deux octets
        loop2_bg1
00804B  8E 18 21     STX $2118             ; mis en $2118 et en $2119 (2 octets)
; $2118 VMDATAL et $2119 VMDATAH = donnée sur 2 octets qui sera écrite dans la VRAM.
00804E  E8           INX                   ; X va donc de #0000 à #003F
; Je ne comprends pas pourquoi cette valeur variable de #00 à #$3F est écrite dans la Vram.
00804F  E0 40 00     CPX.W #$0040          ; = décimal 64 (ni le pourquoi de ce #0040)
008052  D0 F7        BNE loop2_bg1         ; BNE $804B reboucle tant que #0040 n'est pas atteint
008054  60           RTS                   ; done
; Fichtre, l'écran graphique semble alors initialisé ! Le code est clair, mais sa signification m'échappe. Les adresses $7C00 à $7FBFsont écrits avec les valeurs #$0040 à #$033F et les adresses de $7FC0 à $7FFF reçoivent les valeurs de #$0000 à #$003F. Là aussi, c'est à revoir…


; ********************************************************
```

```
        init_charset                    ; Cette routine est uniquement appelée par la routine ″start″ lors du
boot  avec X,Y sur 16 bits et A sur 8 bits. En théorie, mets à zéro la zone des caractères dynamiques et peut-être même
toute la banque $7F. Est-ce un bug ou moi qui ne pige rien?
008055  A2 00 00      LDX.W #0            ; X=#$0000 = sur 2 octets
008058  8A            TXA                 ; A=#00 sur un octet
        charset_clear
008059  9F 00 00 7F   STA $7F0000,X       ; Mets les caractères dynamiques à zéro
00805D  E8            INX                 ; caractères suivants (X va de #0000 à #FFFF)
00805E  D0 F9         BNE charset_clear   ; BNE $8059, reboucle tant que X ne repasse pas par zéro
008060  60            RTS                 ; done
; Je pense que cette routine initialise plus que la zone des 256 caractères dynamiques (ou même que la zone des 1024
caractères possibles), mais bel et bien toute la banque $7F (banque secondaire).


; ************************************************************

        copyrom                         ; Copie la Rom Basic de la flash vers la banque $7E
; Cette routine n’est appelée que par la routine ″start″ lors du boot, avec X,Y sur 16 bits et A sur 8 bits et DBR sur $7E.
008061  A2 00 00      LDX.W #0            ; X=#$0000
        romloop
008064  BF 31 82 00   LDA.L Basic,X       ; LDA $008231,X où $8231 est l'offset du Basic
008068  9D 00 C0      STA $C000,X         ; Copie la Rom Basic de $7EC000 à $7EFFFF
; Le Registre de Banque de Données (DBR) vient d'être initialisé à $7E par la routine start
00806B  E8            INX                 ; X va de #$0000 à #$3FFF
00806C  E0 00 40      CPX #$4000          ; on copie #4000 octets
00806F  D0 F3         BNE romloop         ; BNE $8064, reboucle tant que X n’a pas atteint #4000
008071  60            RTS                 ; done


; ************************************************************

; Handlers d'interruption               ; Attention, il faut passer sur 16 bits avant de sauver les registres,
sinon il se pourrait qu'on ne sauve que des registres 8 bits et qu'on perde la partie haute en passant ensuite sur 16 bits.

        nmi_handler                     ; Concerne les tâches à accomplir en cas d'interruption NMI
; Cette routine n’est pas appelée directement, mais au travers des vecteurs en $FFEE et en $FFFE.
008072  8B            PHB                 ; empile le registre de banque de données (DBR)
008073  4B            PHK                 ; empile le registre de banque de programme (PBR)
008074  AB            PLB                 ; initialise DBR à 0 pour accéder aux IOs
; Je crois comprendre que PBR = $00. On empile cette valeur puis on la dépile dans DBR. La valeur initiale de DBR
reste sur la pile.
008075  08            PHP                 ; empile le registre d'état (Status Register SR)
008076  C2 30         REP #$30            ; A/M X/Y sur 16 bits
008078  48            PHA                 ; pour sauver les registres complets A, X et Y
008079  DA            PHX                 ;
00807A  5A            PHY                 ;
00807B  E2 30         SEP #$30            ; A/M X/Y sur 8 bits
00807D  AE 10 42      LDX $4210           ; reset NMI interrupt (remise à 0 par simple lecture en $004210)
008080  9C 0F 21      STZ $210F           ; BG 2 Horizontal Scroll Offset
008083  9C 0F 21      STZ $210F           ; BG 2 Horizontal Scroll Offset. C'est un truc normal:
; En écrivant dans le registre 2 fois de suite, ce dernier est paramétré dans l'ordre: bits bas (0 à 7) et bits haut (8 à 12).
008086  20 0C 81      JSR dma_palette     ; JSR $810c DMA transfert palette vers Ram Graphique
008089  20 36 81      JSR dma_screen      ; JSR $8136 DMA transfert écran Texte vers VRAM
00808C  20 DA 80      JSR dma_charset     ; JSR $80DA DMA transfert caractères dynamiques vers VRAM
00808F  C2 30         REP #$30            ; A/M X/Y sur 16 bits
008091  7A            PLY                 ; pour récupérer les registres complets Y, X, A
008092  FA            PLX                 ;
008093  68            PLA                 ;
008094  28            PLP                 ; dépile le registre d'état (Status Register SR)
008095  AB            PLB                 ; dépile le registre de banque de données (DBR)
008096  40            RTI                 ;


; ************************************************************
```

```
        irq_handler                     ; Concerne les tâches à accomplir en cas d'interruption IRQ
; Cette routine n'est pas appelée directement, mais au travers des vecteurs en $FFEA et en $FFFA.
008097  E2 30         SEP #$30         ; A/M X/Y sur 8 bits
008099  8B            PHB              ; empile le registre de banque de données (DBR)
00809A  4B            PHK              ; empile le registre de banque de programme (PBR)
00809B  AB            PLB              ; initialise DBR à 0 pour accéder aux IOs
; Je crois comprendre que PBR = $00. On empile cette valeur puis on la dépile dans DBR. La valeur initiale de DBR reste sur la pile.
00809C  48            PHA              ; empile l'accumulateur
00809D  AD 11 42      LDA $4211        ; reset IRQ (remise à zéro par simple lecture en $4211)
0080A0  A9 C0         LDA #$C0         ;
0080A2  8D 0D 03      STA $030D        ; simule une interruption Timer 1
0080A5  68            PLA              ; dépile l'accumulateur
0080A6  AB            PLB              ; dépile le registre de banque de données (DBR)
0080A7  40            RTI              ;

; ********************************************************

        clear_vram                      ; Remet à zéro la Vram (si on peut dire !)
; Cette routine est seulement appelée par la routine "start" lors du boot, avec X,Y sur 16 bits et A sur 8 bits.
0080A8  08            PHP              ; Empile le registre d'état (Status Register SR)
0080A9  C2 30         REP #$30         ; A/M X/Y sur 16 bits
0080AB  A2 00 00      LDX.W #$0000     ; X=#$0000. La notation ldx.w est destinée à l'assembleur.
; En fait, il s'agit d'un bel exemple de différence entre code 6502 (X sur 8 bits) et 65816 (ici X sur 16 bits). Même mnémonique officiel ldx, même opcode #A2, mais sur 2 octets dans un cas et sur 3 octets dans l'autre !
0080AE  8E 16 21      STX $2116        ; Address for VRAM Read/Write. Autre exemple de différence
; entre code 6502 (X sur 8 bits) et 65816 (ici X sur 16 bits). Même mnémonique officiel stx, même opcode  #8E, mais sur 2 octets dans un cas et sur 3 octets dans l'autre ! Curieusement, il n'est pas nécessaire d'indiquer stx.w pour l'assembleur.
; $2116 VMADDL et $2117 VMADDH = adresse initiale pour lire ou écrire sur la VRAM. A chaque fois que la donnée est écrite ou lue dans la VRAM, l'adresse est incrémentée automatiquement. La valeur d'incrémentation est dans le registre $2115, qui n'ayant pas été mis à jour dans cette routine, date de l'une des routines précédemment appelées, par exemple dma_screen ou hires_transfer ou init_ppu (situation malsaine).
0080B1  A9 00 00      LDA.W #0         ; A=#$0000 Pour mettre à zéro la VRAM.
; La notation ldx.w est destinée à l'assembleur. Même remarque que pour ldx et stxx ci-dessus.
        vram_init
0080B4  8D 18 21      STA $2118        ; Ecrit ce #$0000 en $2118/2119 = Data for VRAM Write
; $2118 VMDATAL et $2119 VMDATAH = les données de l'arrière plan et d'objet.  Elles peuvent êtres écrites dans n'importe quelle zone de la VRAM.
0080B7  E8            INX              ; X=#$0000 au départ, incrémenté jusqu'à X=#$7FFF
0080B8  10 FA         BPL vram_init    ; BPL $80B4, reboucle jusqu'à ce que X devienne négatif,
; c'est-à-dire atteigne la valeur #$8000 (1000 0000 en binaire)
0080BA  28            PLP              ; dépile le registre d'état (Status Register SR)
0080BB  60            RTS              ; done
; Contrairement à ce qui se passait pour les autres routines d'initialisation de la Vram (voir ),  ce qui est fait ici est clair. On initialise #$8000 octets de la Vram (16 ko, soit la moitié du total, celle qui correspond aux "données d'arrière-plan et d'objet"), de $0000 à $7FFF avec la valeur #$0000, soit deux octets à la fois. La valeur d'incrémentation, dans le registre $2115,  devrait donc être appropriée (de la forme #$0X, mais j'ignore la valeur de ce X).

; ********************************************************

        init_palette                    ; "initialisation des couleurs"
; Cette routine n'est appelée que par la routine "start" lors du boot, avec X,Y sur 16 bits et A sur 8 bits. Elle met à zéro la zone des palettes en banque $7F, soit de $7FF800 à $7FF9FF, sauf $7ff802 et $7ff803 qui sont mis à #FF.
0080BC  08            PHP              ; empile le registre d'état (Status Register SR)
0080BD  E2 30         SEP #$30         ; A/M X/Y sur 8 bits
0080BF  A2 00         LDX #$00         ;
0080C1  A9 00         LDA #$00         ;
        init_colors
0080C3  9F 00 F8 7F   STA $7FF800,X    ; Palette de $7FF800 à $7FFA00
0080C7  9F 00 F9 7F   STA $7FF900,X    ; Pour écrire deux pages à la fois
0080CB  E8            INX              ; X de #$00 à   #$FF = une page
```

```
0080CC  D0 F5        BNE init_colors          ; BNE $80c3
0080CE  A9 FF        LDA #$FF                 ;
0080D0  8F 02 F8 7F  STA $7FF802              ; sauf $7ff802 et $7ff803 qui sont mis à #FF
0080D4  8F 03 F8 7F  STA $7FF803              ;
0080D8  28           PLP                      ; dépile le registre d'état (Status Register SR)
0080D9  60           RTS                      ; done
; La couleur n°0 ($7ff800-$7ff801) est mise à #0000 (PAPER = noir).
; La couleur n°1 ($7ff802-$7ff803) est mise à #FFFF (INK = blanc).


; ************************************************************

       dma_charset                            ;  Transfert des caractères dynamiques de la banque secondaire
; de $7F000 à $7F0FFF vers la Ram vidéo (Vram). Cette routine est appelée seulement par la routine ″nmi_handler″
avec A, X, Y sur 8 bits et DBR sur $00.
0080DA  08           PHP                      ; empile le registre d'état (Status Register SR)
0080DB  C2 30        REP #$30                 ; A,X,Y sur 16 bits
0080DD  E2 10        SEP #$10                 ; X/Y sur 8 bits
0080DF  A2 80        LDX #$80                 ; Soit 1000 0000 en binaire
; b3 à b0 = 0 pour incrémenter l'adresse, un octet par un octet et b7 = 1 pour incrémenter après que les données soient
écrites dans le registre $2119 ou lues dans le registre $213A.
0080E1  8E 15 21     STX $2115                ; Vram Address Increment Value after data written to $2119
0080E4  A9 00 00     LDA.W #$0000             ; Pour écrire en Vram à partir de l'adresse $0000
0080E7  8D 16 21     STA $2116                ; Address for VRAM Read/Write
; $2116 VMADDL et $2117 VMADDH = Adresse pour la lecture et l'écriture en Vram. L'adresse réelle de la Vram est
transparente. A partir du moment où on utilise le registre $2116-$2117  (Address for Vram Write), le système sait où il
doit transférer.
0080EA  A2 01        LDX #$01                 ; Soit 0000 0001 en binaire. Initialise les paramètres pour le
; transfert de données par DMA. On a b7 à 0 pour  BUS A vers BUS B (Mémoire CPU vers PPU), b3 et b4 à 0 pour
Incrémentation automatique et b0 =1, pour écriture de deux octets à chaque fois.
0080EC  8E 00 43     STX $4300                ; $43K0 Parameters for DMA Transfer (ici K=0 pour canal 0)
0080EF  A2 18        LDX #$18                 ; Octet faible pour former l'adresse du BUS B
0080F1  8E 01 43     STX $4301                ; en VRAM (donc résultat = $002118)
; $43X1 adresse du BUS B pour le DMA (X = numéro de canal : 0 à 7)
0080F4  A9 00 00     LDA.W #$0000             ; Pour assembler adresse $7F0000 des caractères dynamiques
0080F7  8D 02 43     STA $4302                ; A Address (sur deux octets en $4302/4303)
0080FA  A2 7F        LDX #$7F                 ; Banque secondaire Oric
0080FC  8E 04 43     STX $4304                ; A Address Bank (au total $7F0000)
0080FF  A9 00 10     LDA.W #$1000             ; Pour copier #1000 octets de $7F000 à $7F0FFF
008102  8D 05 43     STA $4305                ; Number Bytes to Transfer
; $43X5 = registre pour paramétrer le nombre d'octets à être transférés.
008105  A2 01        LDX #$01                 ;
008107  8E 0B 42     STX $420B                ; Lance le transfert
00810A  28           PLP                      ; dépile le registre d'état (Status Register SR)
00810B  60           RTS                      ;


; ************************************************************

       dma_palette                            ; Transfert par DMA de la palette de la 2e banque Oric, de
; $7FF800 à $7FFA00, vers la Ram Graphique (CG-RAM). Cette routine est appelée seulement par la routine
″nmi_handler″ avec A, X, Y sur 8 bits et DBR sur $00.
00810C  08           PHP                      ; Sauve le registre d'état (Status Register SR)
00810D  C2 30        REP #$30                 ; A/M X/Y sur 16 bits
00810F  E2 10        SEP #$10                 ; X/Y sur 8 bits
008111  A2 00        LDX #$00                 ; commencer à la couleur 0 = au début de la palette
008113  8E 21 21     STX $2121                ; Address for CG-RAM Write (de $00 à $FF pour 256 couleurs)
; $2121 = CGADD adresse initiale pour lire/écrire dans la CG-RAM
008116  8E 00 43     STX $4300                ; $43K0 Parameters for DMA Transfer (ici K=0 pour canal 0)
; X=0 donc b7 à 0 = BUS A vers BUS B (Mémoire CPU vers PPU), b3 et b4 à 0 = Incrémentation automatique,
et b0 = 0, soit écriture d'un octet à la fois.
008119  A2 22        LDX #$22                 ; Octet faible pour former l'adresse du BUS B
00811B  8E 01 43     STX $4301                ; B Address, ici $002122
; $43K1 adresse du BUS B pour le DMA (K = numéro de canal de 0 à 7, ici canal 0)
```

```
00811E  A9 00 F8     LDA #$F800          ; L'adresse source de la palette est $7FF800
008121  8D 02 43     STA $4302           ; A Address (sur deux octets en $4302/4303)
008124  A2 7F        LDX #$7F            ; Banque secondaire Oric
008126  8E 04 43     STX $4304           ; A Address Bank
008129  A9 00 02     LDA #$0200          ; Pour transférer les 512 octets de la palette
00812C  8D 05 43     STA $4305           ; Number Bytes to Transfer (2 octets par couleur)

; $43K5 et $43K6 (K = numéro de canal : de 0 à 7, ici canal 0) Nombre d'octets à être transférés (ici #$200).
00812F  A2 01        LDX #$01            ;
008131  8E 0B 42     STX $420B           ; Lance le transfert de A vers B ($7FF800 vers CG-RAM)
008134  28           PLP                 ; dépile le registre d'état (Status Register SR)
008135  60           RTS                 ; done
; Note : L'adresse réelle de la CG-ram est transparente. A partir du moment où on utilise le registre $2121 (Address for CG-RAM Write), le système sait où il doit transférer.


; ************************************************************

        dma_screen                      ; Transfert par DMA de l'écran Texte (2e banque Oric,
; de $7FF000 à $7FF800, vers Vram. Cette routine est appelée seulement par la routine "nmi_handler" avec A, X, Y sur 8 bits et DBR sur $00.
008136  08           PHP                 ; empile le registre d'état (Status Register SR)
008137  C2 30        REP #$30            ; A/M X/Y sur 16 bits
008139  E2 10        SEP #$10            ; X/Y sur 8 bits
00813B  A2 80        LDX #$80            ; Modalités d'incrémentation de l'adresse en $2116-$2117
; pour écrire dans le registre $2118-$2119 ou lire dans le registre $2139-$213A (modalités pour le moins obscures).
00813D  8E 15 21     STX $2115           ; VMAINC Vram Address Increment Value
008140  A9 00 78     LDA #$7800          ; Pour écrire en Vram à partir de l'adresse $7800
008143  8D 16 21     STA $2116           ; Address for Vram Read/Write
; $2116 VMADDL et $2117 VMADDH = Adresse pour la lecture et l'écriture en Vram. L'adresse réelle de la Vram est transparente. A partir du moment où on utilise le registre $2116-$2117  (Address for Vram Write), le système sait où il doit transférer.
008146  A2 01        LDX #$01            ; Soit 0000 0001 en binaire. Initialise les paramètres pour le
; transfert de données par DMA. On a b7 à 0 pour  BUS A vers BUS B (Mémoire CPU vers PPU), b3 et b4 à 0 pour Incrémentation automatique et b0 =1, pour écriture de deux octets à chaque fois.
008148  8E 00 43     STX $4300           ; $43K0 Parameters for DMA Transfer (ici K=0 pour canal 0)
; X=1 donc b0 =1, soit écriture de deux octets à chaque fois et b7=0 pour transfert BUS A vers BUS B.
00814B  A2 18        LDX #$18            ; Initialise l'adresse du BUS B à $002118-$002119 (écriture)
00814D  8E 01 43     STX $4301           ; $43K1 = adresse du BUS B pour le DMA (ici K=0 canal 0)
008150  A9 00 F0     LDA #$F000          ; $43K2, $43K3 et $43K4 forment l'adresse du BUS A
008153  8D 02 43     STA $4302           ; A Address (sur deux octets en $4302/4303)
008156  A2 7F        LDX #$7F            ; Pour adresse de l'écran texte $7FF000 à $7FF800
008158  8E 04 43     STX $4304           ; A Address Bank, ici $7F 2e banque Oric
00815B  A9 00 08     LDA #$0800          ; pour transférer les #800 octets de l'écran texte
00815E  8D 05 43     STA $4305           ; Number Bytes to Transfer (DMA)
; $43K5 et $43K6 forment le nombre d'octets à transférer, ici K=0 pour canal 0.
008161  A2 01        LDX #$01            ; Pour démarrer le transfert du DMA du canal 0
008163  8E 0B 42     STX $420B           ; Regular DMA Channel Enable
008166  28           PLP                 ; dépile le registre d'état (Status Register SR)
008167  60           RTS                 ; done


; ************************************************************

        hires_transfer                  ; Transfert par DMA de l'écran Hires (2e banque Oric) vers la
; VRAM. Cette routine est appelée uniquement à partir du Basic (REFRESH), via le vecteur en $008004
008168  08           PHP                 ; Sauve le registre d'état (Status Register SR)
008169  8B           PHB                 ; Sauve le registre de banque de données (DBR)
00816A  4B           PHK                 ; Empile le registre de banque de programme (PBR = $00)
00816B  AB           PLB                 ; Dépile dans le registre de banque de données (met DBR = $00)
00816C  C2 30        REP #$30            ; A/M X/Y sur 16 bits
00816E  E2 10        SEP #$10            ; X/Y sur 8 bits
008170  48           PHA                 ; Sauve les 16 bits de A
008171  A2 80        LDX #$80            ; Pour écran éteint (b7 à 1) afin de permettre le transfert
```

```
008173  8E 00 21      STX $2100               ; INIDISP Screen Display Register
; Tous les registres $21xx sont en Ram dans la banque $00 et ses répliques ($00 à $3F et $80 à $BF)
008176  A2 8C         LDX #$8C                ; Modalités d'incrémentation de l'adresse en $2116-$2117
; Pour écrire dans le registre $2118-$2119 ou lire dans le registre $2139-$213A (modalités pour le moins obscures).
008178  8E 15 21      STX $2115               ; VMAINC VRAM Address Increment Value
00817B  A9 00 08      LDA #$0800              ; Adresse initiale dans la Vram sur 16 bits
00817E  8D 16 21      STA $2116               ; VMADDL Address for VRAM Read/Write (Low Byte)
```

; $2116 VMADDL et $2117 VMADDH = Adresse initiale pour lire ou écrire sur la Vram, ici $0800. A chaque fois que la donnée est écrite ou lue dans la VRAM, l'adresse sera incrémentée automatiquement. La valeur d'incrémentation est déterminée par le registre $2115 (voir ci-dessus). On ne sait pas à quoi correspond cette adresse $0800. L'adresse réelle de la Vram est transparente. A partir du moment où on utilise le registre $2116-$2117 (Address for Vram Write), le système sait où il doit transférer. A la limite, il n'est pas important de le savoir, puisque tout est interfacé par les registres $2118-$2119 (pour écrire) $2139-$213A (pour lire).

```
008181  A2 01         LDX #$01                ; Initialise paramètre pour transfert de donnée par DMA
; X=0000 0001, avec  b7 à 0 = BUS A vers BUS B (Mémoire CPU vers PPU),
; b3 et b4 à 0 = Incrémentation automatique et b0 à1 =  écriture de deux octets à chaque fois.
008183  8E 00 43      STX $4300               ; $43K0 = Parameters for DMA Transfer (ici K=0 pour canal 0)
; Tous les registres $42xx et $43xx sont en Ram dans la banque $00 et ses répliques ($00 à $3F et $80 à $BF)
008186  A2 18         LDX #$18                ; Initialise l'adresse du BUS B à $2118-$2119 (adresse cible)
008188  8E 01 43      STX $4301               ; $43K1 = Adresse du BUS B pour le DMA (ici K=0 pour canal 0)
00818B  A9 00 10      LDA #$1000              ; $43K2, $43K3 et $43K4 forment l'adresse du BUS A
00818E  8D 02 43      STA $4302               ; A Address (sur deux octets en $4302/4303)
008191  A2 7F         LDX #$7F                ; Pour adresse de l'écran Hires $7F1000 à $7FF000
008193  8E 04 43      STX $4304               ; A Address Bank, ici $7F banque secondaire Oric
008196  A9 00 E0      LDA #$E000              ; Initialise le nombre d'octets à transférer, ici #E000
008199  8D 05 43      STA $4305               ; Number Bytes to Transfer
; Les registres $43K5 et $43K6 contiennent le nombre d'octets à transférer, ici K=0 pour canal 0.
00819C  A2 01         LDX #$01                ;
00819E  8E 0B 42      STX $420B               ; Pour lancer le transfert du DMA du canal 0
0081A1  A2 0F         LDX #$0F                ; Ecran allumé (b7 à 1), brillance maximale (b0 à b3 à 1)
0081A3  8E 00 21      STX $2100               ; INIDISP Screen Display Register
0081A6  68            PLA                     ; récupère le registre A (16 bits)
0081A7  AB            PLB                     ; récupère le registre de banque de données (DBR)
0081A8  28            PLP                     ; récupère le registre d'état (Status Register SR)
0081A9  60            RTS                     ; done


; ********************************************************

        init_ppu                              ; (Picture Processing Unit)
; Cette routine est seulement appelée par la routine "start" lors du boot, avec X,Y sur 16 bits et A sur 8 bits.
0081AA  08            PHP                     ; empile le registre d'état (Status Register SR)
0081AB  C2 30         REP #$30                ; A/M X/Y sur 16 bits
0081AD  0B            PHD                     ; Push Direct Page Register
0081AE  A9 00 21      LDA.W #$2100            ; A=#$2100
; Les registres du PPU vont de $2100 à $21FF, espace qui devient la "direct page" DP
0081B1  5B            TCD                     ; DP en $2100 pour accéder aisément aux IOs
; Transfer 16-bit Accumulator to Direct Page Register
0081B2  E2 20         SEP #$20                ; A/M sur 8 bits
0081B4  A9 80         LDA #$80                ;
0081B6  85 00         STA $00                 ; screen off (b7 = 1 mis en $2100)
0081B8  64 01         STZ $01                 ; OBJSEL mis à zéro
; $2101 = OBJSEL = Taille de l'objet et paramétrage de la zone de donnée de l'objet
0081BA  A9 04         LDA #$04                ; Soit 0000 0100 en binaire, b3 = 0 priorité, b2-b0 = 4 mode
0081BC  85 05         STA $05                 ; MODE 4 : BG2 tuiles en 4 couleurs
; Plus précisément, Mode 4 = BG1 (Hires) 256 couleurs, 1 palette et BG2 (Text) 4 couleurs, 8 palettes
0081BE  64 06         STZ $06                 ; $2106 mis à zéro = mosaic off
0081C0  A9 7C         LDA #$7C                ; soit 0111 1100 en binaire
; Base d'adresse d'arrière-plan (6 bits haut), désigne le segment dans lequel les données dans la Vram sont stockées (1K-Word/segment).
0081C2  85 07         STA $07                 ; BG1 = Vram $7C00-7fff (32x28)
0081C4  A9 78         LDA #$78                ; soit 01111000 en binaire
0081C6  85 08         STA $08                 ; BG2 = Vram $7800-7bff (32x28)
```

```
0081C8  A9 00        LDA #$00          ; BG1 tuiles = $0800-$77ff et BG2 tuiles = $0000-$07ff (128 tuiles)
0081CA  85 0B        STA $0B                   ; on utilise les tuiles BG1 à partir de 64 pour commencer en $800
0081CC  A9 FF        LDA #$FF                  ;
0081CE  64 0D        STZ $0D                   ;
0081D0  64 0D        STZ $0D                   ; 0 dans BG1HOFS en $210D
; En écrivant dans le registre deux fois de suite, ce dernier est paramétré dans l'ordre: bits bas (0 à 7) et bits haut (8 à 15). Idem pour les registres suivants.
0081D2  85 0E        STA $0E                   ;
0081D4  85 0E        STA $0E                   ; -1 dans BG1VOFS en $210E
; $210D BG1H0FS et $210E BG1V0FS = défilement Horizontal et Vertical pour BG-1.
0081D6  64 0F        STZ $0F                   ;
0081D8  64 0F        STZ $0F                   ; 0 dans BG2HOFS en $210F
0081DA  85 10        STA $10                   ;
0081DC  85 10        STA $10                   ; et -1 dans BG2VOFS en $2110
; $210F BG2HOFS et $2110 BG2VOFS = défilement Horizontal et Verticale pour BG-2
0081DE  A9 80        LDA #$80                  ; soit 1000 0000 en décimal
0081E0  85 15        STA $15                   ; accès word pour la Vram en $2115
; $2115 VMAINC = désignation de la valeur incrémentée par l'adresse VRAM. b7=1 l'adresse sera incrémentée après que les données soient écrites dans le registre $2119 ou lues dans le registre $213A.
0081E2  64 23        STZ $23                   ; window off $2123 W12SEL pour BG1 et BG2
0081E4  64 24        STZ $24                   ; window off $2124 W34SEL pour BG3 et BG4
0081E6  64 25        STZ $25                   ; window off $2125 WOBJSEL pour objet
0081E8  A9 02        LDA #$02                  ; enable BG2
0081EA  85 2C        STA $2C                   ; in main screen, écran de premier plan = BG2
0081EC  64 2D        STZ $2D                   ; no sub screen, pas d'écran de second plan
0081EE  64 2E        STZ $2E                   ; no window mask in main screen
0081F0  64 2F        STZ $2F                   ; no window mask in sub screen
0081F2  64 30        STZ $30                   ; no color addition $2130 CGSWSEL en mode direct select
0081F4  64 31        STZ $31                   ; $2131 CGADSUB = param Addition/Soustraction couleur
0081F6  64 32        STZ $32                   ; $2132 COLDATA = donnée pour l'Addition/Soustraction
0081F8  64 33        STZ $33                   ; $2133 SETINI = paramétre initial de l'écran.
0081FA  2B           PLD                       ; Pull Direct Page Register, restaure valeur initiale
0081FB  9C 01 42     STZ $4201                 ; Programmable I/O Port Output, ici mis en sortie
0081FE  A9 81        LDA #$81                  ;
008200  8D 00 42     STA $4200                 ; NMI, V/H Count, and Joypad Enable
; $4200 NMITIMEN Active l'indicateur pour V-BLANK, interruption du timer et lecture du contrôleur (la manette) standard. Ici b7=1 NMI Activé. b5 et b4 nuls V/H Count désactivé. b0=1 active la lecture automatique du contrôleur standard.
008203  28          PLP                       ; dépile le registre d'état (Status Register SR)
008204  60          RTS                       ; done


; ********************************************************

       init_charset_16_colors                 ; Cette routine n'est appelée de nulle part.
; D'ailleurs, le mode Lores1 (caractères définis en 16 couleurs) n'a été implémenté que beaucoup plus tard. C'est donc une routine à garder sous le coude, d'autant quelle est  intéressante. Dans le mode Lores1, on peut toujours définir 1024 caractères, mais seulement les 128 premiers sont dynamiques. Ceux-ci sont définis dans la zone $7F0000-$7F0FFF à raison de 32 octets par caractères. Notons que pour définir 1024 caractères en 16 couleurs, il faut 1024 x 32 = 32768 octets (#8000), de $7F0000 à $7F7FFF ce qui empiète sur la moitié de l'écran Hires. Conclusion, en mode Lores1, il faut choisir entre 128 caractères maxi avec Hires ou définir plus de 128 caractères sans Hires.
008205  08          PHP                       ; empile le registre d'état (Status Register SR)
008206  C2 30        REP #$30                  ; A/M X/Y sur 16 bits
008208  E2 20        SEP #$20                  ; A/M sur 8 bits
00820A  A2 00 00     LDX.W #0                  ; X=#$0000
00820D  A0 00 00     LDY.W #0                  ; Y=#$0000
       charset16_loop
008210  B9 A9 BE     LDA Basic+$3C78,Y         ; soit ici LDA $BEA9,Y
; Dans la formule "Basic+$3C78", l'adresse du Basic dans la banque $00 est $8231 (après les #$231 octets de code 65816). La définition des caractères Oric se trouve en Rom, de $FC78 à $FF77 soit #300 octets. Donc l'offset de ces caractères dans le fichier Rom (taille $4000 octets) est $3C78. Au total, on a donc $8231 + $3C78 = $BEA9.
; Notons que le jeu des 96 caractères de l'Oric classique va de l'Ascii 32 (espace) à l'Ascii  127(bloc 6x8). Définis en 2 couleurs, soit 8 octets par caractère, ils occupent 96 x 8 = 768 octets (#300) en mémoire vive. Transposés au Super-Oric
```

en Lores0 (4 couleurs), ce jeu occupe 96 x 16 = 1536 octets (#600 octets). Mais dans la zone des "Caractères dynamiques", les définitions de caractères sont rangées en fonction de leur code Ascii et les 32 premiers caractères (Ascii 0 à 31) sont redéfinissables et leur place est réservée au début de la zone des "Caractères dynamiques". La place nécessaire est donc de 128 x 16 = 2048 octets (#800), allant de $7F0000 à $7F07FF. Dans cette zone des "Caractères dynamiques", il reste donc de la place pour définir 32 caractères de Ascii 0 à 31 et 32 caractères de Ascii 128 à 160. Après, on emprunte sur l'écran Hires.
; Dès l'origine de la Rom Super-Oric, Fabrice a envisagé d'implémenter le mode Lores1. Mais il semble avoir remis à plus tard la finalisation de la présente routine, sensée installer dans la banque $7F les définitions du jeu de caractères Oric en 16 couleurs. Telle qu'elle est actuellement, cette routine initialise un jeu en 4 couleurs.

```
008213  9F 00 02 7F   STA $7F0200,X          ;
```

; Dans la zone caractères dynamiques dans la banque secondaire Oric. Sachant que le jeu normal commence avec l'Ascii 32 (espace) et que chaque caractère est défini sur 16 octets, la copie commence avec un offset de 32x16=512 soit #200, donc en $7F0200. Mais ceci n'est valable qu'en Lores0.

```
008217  49 FF         EOR #$FF              ; Inversion des points pour affichage vidéo inverse
008219  9F 00 0A 7F   STA $7F0A00,X         ; Une copie de chaque caractère dont tous les bits ont été inversés
```

; est également écrite de $7F0A00 à $7F1000, probablement en vue d'un affichage en vidéo inverse. Il reste donc deux "bandes" de libres dans la zone des caractères dynamiques: La première de $7F0000 à $7F01FF (Ascii 0 à 31), la seconde (sorte d'homologue inverse) de $7F0800 à $7F09FF.

```
00821D  E8            INX                   ; Comme il faut passer de 8 octets par caractère à 8 words  par
```

; caractère, un octet de poids fort à #00 sera ajouté à chaque octet pour former un word.

```
00821E  A9 00         LDA #$00              ; pour l'octet de poids fort
008220  9F 00 02 7F   STA $7F0200,X         ; pour le caractère normal
008224  9F 00 0A 7F   STA $7F0A00,X         ; pour le caractère inversé
008228  E8            INX                   ; octet suivant dans la banque $7F
008229  C8            INY                   ; octet suivant dans la Rom basic
00822A  C0 00 03      CPY.W #$0300          ; il y 8x96 caractères = 768 octets à copier, soit #300
00822D  D0 E1         BNE charset16_loop    ; BNE $8210
00822F  28            PLP                   ; dépile le registre d'état (Status Register SR)
008230  60            RTS                   ;


; ************************************************************

        Basic                               ; 16 Ko donc de $008231 à $00C230
008231  BIN BASIC2.ROM                      ; insérer ici le fichier Basic2.rom


; ************************************************************
```

; Et pour terminer, quelques vecteurs dont la Snes a besoin pour fonctionner. Le copyright sera automatiquement ajouté en $7FC0 en fonction du pseudo mnémonique NAM placé tout au début de ce fichier source.

```
        PAD $FFE6                           ; Mettre ce qui suit en $FFE6
        DCW $0000                           ; brk_handler non initialisé ($C10E dans v1.46)
        DCW $0000                           ; abort_handler non initialisé ($C10D dans v1.46)
        DCW nmi_handler                     ; $8072
        DCW $8008                           ; $8008
        DCW irq_handler                     ; $8097

        PAD $FFFA                           ; Mettre ce qui suit en $FFFA
        DCW nmi_handler                     ; $8072
        DCW $8000                           ; $8000
        DCW irq_handler                     ; $8097
```