Voici un résumé du contenu de la première Rom Super-Oric : la Rom Base1.Bin

A) De quoi s'agit-il?

C'est la Rom Super-Oric présente dans les deux premières cartouches Super-Oric:

a) Cartouche SuperOric.swc du 10/06/2002, taille #40200 octets, CRC#5F418586 et CS#EAEA (Entête #200 octets, Rom Base1.bin, programme ″Demo graphique″, 60 tableaux Sokobac, Palette, Image). La partie ″Sokobac″ est inutilisée.

b) Cartouche SuperSoko.swc du 10/06/2002, taille #40200 octets, CRC#DF33A934 et CS#F0BA (Entête #200 octets, Rom Base1.bin, 60 tableaux Sokobac, Palette, Image).

Les éléments contenus dans ces cartouches figurent entre parenthèses. On y voit Base1.bin, qui correspond à la Rom Super-Oric proprement dite. Voici les caractéristiques de cette Rom v0.01: Taille #8000 octets, CRC#EB7BAF7D et CS#FDBE. Les #4000 octets de la partie Basic ont CRC de #D0165863. Nous verrons en quoi consistent les #4000 octets restants.

B) Pourquoi s'intéresser à la Rom Super-Oric v0.01?

1) Parce que c'est celle qui correspond le mieux à l'article princeps de Fabrice (Ceo-Mag n°251, pages 14-17). Il y a cependant quelques exceptions (la commande '&' dans la Rom 0.01 correspond à 'USR' dans l'article, Lores 1 n'existe pas encore dans la Rom 0.01, etc.).

2) Parce que cette première tentative est de loin la plus simple (le "code 65816" par exemple n'occupe que #231 octets). De nouvelles commandes (Lores 1, Son, clavier, etc.) ont été ajoutées peu à peu dans les versions ultérieures.

3) Parce c'est la seule version qui affiche correctement une image haute résolution. Nous essaierons de voir pourquoi.

C) Structure des Roms Super-Oric

Alors que la taille de la Rom Atmos est de #4000 octets, celle du Super-Oric mobilise #8000 octets. On distingue deux parties:

a) Les #4000 octets de la partie Basic, qui correspond à une Rom Atmos v1.1b patchée pour l'adapter au hardware de la Snes. Cette partie est codée dans le fichier source Basic2.asm pour l'assembleur 6502 Frankenstein, utilisé par Fabrice.

b) Les #4000 octets restants sont occupés par une partie "initialisation de la Snes et interfaçage Snes/Oric", écrite en code 65816, d'où l'appellation "code 65816", avec guillemets. Cette partie est codée dans un fichier source Oric2.s pour l'assembleur 65816 Snesasm, utilisé par Fabrice. Dans la version 0.01, elle n'occupe que #231 octets. Les #4000 - #231 octets restants sont occupés par une zone spécifique à la Snes (copyright, vecteurs d'interruptions, etc.) et par des #00 non significatifs.

La partie "Basic"

C'est la plus facile à aborder, d'une part, parce que son homologue Atmos est décrite en détail dans "l'Oric à nu", d'autre part, parce qu'elle est codée en langage machine 6502, que nous pratiquons depuis des lustres. Enfin, je devrais dire "presque entièrement codée en 6502", car la Snes exécute cette Rom Oric en mode natif 65816 (mais avec les registres sur 8 bits). Ceci présente quelques inconvénients, par exemple avec les TXS (voir ci-dessous), mais aussi deux gros avantages:

a) Possibilité d'utiliser des adressages "longs" et donc d'exécuter des routines ou de lire/écrire dans d'autres banques de la Snes.

b) Utiliser des mnémoniques 65816, permettant d'écrire un code beaucoup plus compact (souvent deux opérations sont exécutées avec une seule commande) et donc de patcher plus facilement dans l'espace trop limité des #4000 octets de la Rom Basic.

Vous allez me dire: Comment peut-on écrire du code source 65816 pour un assembleur 6502 ? Et bien c'est l'enfance de l'art, il suffit de "cacher" ce code en entrant directement les valeurs hexadécimales comme si c'était des data. Voici donc les adaptations apportées au Basic:

A) Débogage 6502 pour exécution en 65816.

Cela aurait été trop beau si le mode 65816 était complètement compatible avec le code 6502. Voici un exemple de problème en mode natif 65816, signalé par Fabrice: Le registre S est sur 16 bits et TXS charge aussi le poids fort de X, qui vaut 0 quand le registre X est sur 8 bits. Solution choisie par Fabrice: Remplacer les TXS par des TCS, en gardant toujours la valeur 1 dans le poids fort de l'accumulateur (dans B donc). Une douzaine de mini-patchs ont ainsi été nécessaires. Par la suite d'autres problèmes, jusqu'ici non repérés, sont apparus, qui ont fait l'objet de débogages spécifiques dans les versions ultérieures. Bravo à Fabrice d'avoir analysé le pourquoi du comment et bouché les quelques "trous noirs" (non ou mal documentés) du passage 6502 en 65816! Nous verrons plus loin, que la nécessité d'utiliser des adressages longs (sur 3 octets au lieu de 2) a rendu encore plus tendu le problème du manque de place dans la Rom Basic.

B) Modification du nom de certaines commandes Basic.

Cette Rom est la seule à utiliser les nouvelles commandes REFRESH, LOAD, SAVE et JOY: En fait quatre commandes ont été renommées:

Token	Commande classique		Nouvelle commande
#A0	RELEASE	->	REFRESH
#B6	CLOAD	->	LOAD
#B7	CSAVE	->	SAVE
#DD	&	->	JOY

Par exemple, dans un listing Super-Oric, il faut écrire REFRESH, mais un LIST sous Atmos donne RELEASE. En effet REFRESH (qui est le mot-clef compris par l'interpréteur Super-Oric) est codé par le token #A0, mais lors d'un LIST sur Atmos, c'est RELEASE qui est affiché! Les mots-clefs REFRESH, LOAD, SAVE, et JOY ne sont pas compris par l'interpréteur de l'Atmos, ni par Txt2bas.exe, d'où une difficulté pour obtenir un fichier ".tap", quand on ne dispose pas de clavier pour Super-Oric.

Le plus simple est de taper un ficher texte avec les mots-clefs REFRESH, CLOAD, CSAVE et & dans leur syntaxe normale. Puis de mouliner avec Txt2bas.exe (option -v1). Le fichier tap résultant est alors utilisable par le Super-Oric, avec la Rom v0.01.

Attention, par la suite et à cause de cette difficulté, Fabrice retourne aux noms de commandes classiques (RELEASE, CLOAD etc.). Il remplace aussi le &/JOY de token #DD par USR de token #D9. Les programmes ".tap" de la première génération ne marcheront plus avec les Roms suivantes (sauf édition manuelle du token #DD en #D9).

Notez encore que le nom de la commande FILL n'a pas été modifié, mais que son rôle a été complètement transformé.

C) Les routines supprimées ou re-attribuées.

Pour adapter le Basic de l'Atmos à la Snes, Fabrice a eu besoin de place. Hors la place est chère dans la Rom 1.1b. Certaines routines sont devenues obsolètes, liées par exemple à la gestion du port cassette. Fabrice a aussi considéré que d'autres routines, comme celle de génération des caractères alternés ne sont plus vraiment utiles (puisqu'on peut redéfinir maintenant jusqu'à 1024 caractères) et ont été remplacées par autre chose.

D) Les commandes supprimées ou re-attribuées.

Il s'agit de CHAR, CLOAD, CSAVE, FILL, GRAB, RELEASE et & (qui deviendra USR dans les Roms ultérieures). Ces commandes sont devenues obsolètes:

1) GRAB et RELEASE (puisque dans le Super-Oric, les écrans ont été déportés dans la banque secondaire). La place libérée a été consacrée au rafraîchissement de la Ram vidéo (commande REFRESH).

2) CLOAD et CSAVE (puisqu'il n'existe plus de port cassette). La place libérée a été ré-affectée à la lecture/écriture en Rom Flash (commandes LOAD et SAVE).

3) FILL (assez curieuse dans le Basic Oric par rapport aux autres Basic). Sur Oric, cette commande servait principalement à insérer des attributs dans un graphique. Avec le Super-Oric, les attributs n'existent plus puisque chaque point peut avoir sa propre couleur. La nouvelle commande FILL retrouve donc un comportement plus usuel, à savoir remplir de couleur une surface délimitée par une couleur de bordure.

4) & (ou USR dans les Roms ultérieures). Le Super-Oric étant équipé par défaut d'un joypad, il fallait disposer d'une commande de lecture de ce joypad. D'ou la ré-affectation du vecteur &.

5) CHAR ne présentait plus d'intérêt, puisque les écrans Txt et Hires coexistent simultanément. Elle a donc été ré-attribuée pour faire DEF CHAR (voir plus loin).

E) Modification de l'adresse d'exécution de certaines commandes Basic.

Bien évidemment, il n'est pas toujours possible de patcher les nouvelles routines dans la place disponible. Il s'ensuit inévitablement quelques changements dans la table des adresses des commandes:

a) En $C046, l'adresse de la commande RELEASE/REFRESH ($EC0C) devient celle de la commande GRAB ($EBE7). La commande REFRESH est utilisée pour rafraîchir la Ram vidéo. Je n'ai pas testé, mais GRAB devrait donc faire la même chose.

b) En $C072, l'adresse de la commande CLOAD/LOAD ($E85B) est remplacée par $E6DA. La commande CSAVE/SAVE est restée en stand-by en attendant l'arrivée de cartouches R/W.

c) En $C076, l'adresse de la commande DEF ($D4BA) devient $F824 à la suite d'un remaniement indispensable.

d) En $C098, l'adresse du vecteur de & ($02FB) devient $F287 (routine joyread).

e) En $EAD1, l'adresse de la commande FILL est passé de $F268 à $F21E.

f) De même en $EAD5, les adresses des commandes PAPER (ex $F204) et INK (ex $F210) sont respectivement devenues $F1C8 et $F1F6.

F) Les commandes et routines adaptées au hardware de la Snes.

Quelques changements importants sont intervenus avec la Snes et concernent:

a) Les dimensions de l'écran Text. La largeur de l'écran Text passe de 40 à 32 colonnes. La disparition des attributs vidéo rend obsolète les deux colonnes réservées à gauche de l'écran. Dans les faits, la largeur de l'écran Text passe donc de 40-2=38 à 32. L'index des abscisses va donc de x=0 à x=31. La ligne service (sans n° sur Atmos) devient la ligne n°0. Les 28 lignes de l'écran vont donc du n°0 au n°27 et (sauf bug) l'index des ordonnées va donc de y=0 à y=27. En conséquence, les routines impliquées dans l'écran Text ont été adaptées. Cela n'a l'air de rien, mais cela touche non seulement les parties de code gérant les index x et y, mais aussi celles gérant la protection de la ligne service et des 2 colonnes de gauche. A l'exception de la commande PRINT de base (sans argument), toutes les commandes impliquées dans l'affichage Text ont accès à l'ensemble de l'écran, aussi bien à la ligne service, que les colonnes de gauche. Cela représente pas mal de patches dans la Rom!

b) Les dimensions de l'écran Hires. Elles passent de 240x200 à 256x224. De plus, l'affichage simultané des écrans Text et Hires superposés rend obsolète la présence des 3 lignes Text au bas de l'écran Hires. Les nouvelles coordonnées de l'écran Hires sont donc x=0 à x=255 et de y=0 à y=223. En conséquence, les routines impliquées dans l'écran Hires ont été adaptées.

c) Les traditionnelles 8 couleurs de l'Oric (de 0 à 7) sont maintenues par défaut, mais peuvent être étendues à 256 couleurs à choisir parmi 32768. Huit palettes peuvent être définies pour l'écran Text. Chaque palette compte 4 couleurs (en Lores 0) ou 16 couleurs (en Lores 1, ce mode n'étant pas encore implémenté dans la Rom 0.01). Une seule palette de 256 couleurs peut être définie pour l'écran Hires, sachant qu'au total (Text + Hires) on ne peut définir plus de 256 couleurs). En conséquence, les routines impliquées dans la gestion des couleurs ont été adaptées.

d) Les traditionnels 96 caractères de l'Oric (de Ascii 32 à Ascii 127) sont maintenues par défaut, mais peuvent être étendus à 1024, dont les 256 premiers sont dits "dynamiques" (les redéfinitions de caractères ont un effet immédiat à l'écran, sans avoir à attendre un rafraîchissement de la Vram). Les attributs d'écran ont disparus. Chaque caractère, codé sur deux octets au lieu d'un seul, possède ses propres attributs. Chaque caractère affiché peut

utiliser une des 8 palettes de 4 ou 16 couleurs, être retourné horizontalement ou verticalement comme dans un miroir et placé devant ou derrière l'écran Hires. En conséquence, les routines impliquées, d'une part dans les attributs d'écran, d'autre part dans la gestion des caractères ont été adaptées.

e) L'espace mémoire, limité à 64 Ko dans l'Oric classique, est doublé. Le Super-Oric utilise deux banques de Ram de 64 Ko chacune: Une banque principale (banque n°$7E de la Snes) et une banque secondaire (banque n°$7F de la Snes). Cette dernière accueille les définitions des caractères, l'écran Hires, l'écran Text et les définitions des couleurs. C'est autant de place dégagée sur la mémoire utilisable par le Basic: Sur les 64 Ko de Ram de la banque principale, le Super-Oric utilise 16 Ko pour l'interprète Basic (qui ne tourne donc plus en Rom, mais en Ram) et il reste maintenant 48 Ko pour les programmes en Basic. En conséquence, les routines impliquées dans la gestion de l'espace mémoire ont été adaptées. Si PEEK et POKE, par exemple fonctionnent comme par le passé, elles ne s'adressent qu'à la banque principale. Aie, Aie, Aie, on ne peut pas lire/écrire directement dans la banque secondaire où se trouvent les écrans Text et Hires, les redéfinitions de caractères et de couleurs. Quand c'était incontournable, Fabrice a donc introduit de nouvelles commandes pour accéder aux données de la banque secondaire. Afin de ne pas introduire de nouveaux mots-clefs (la place manque dans les tables des noms et adresses des mots-clefs) il a utilisé une association de mot-clefs existant:

- DEF INK, qui définit une ou plusieurs couleurs consécutives de la palette.
- DEF CHAR, qui est bien plus simple à utiliser pour définir des caractères que les classiques POKE.
- PLOT qui permet d'afficher un caractère (code Ascii + attributs) à une position donnée de l'écran Text.
- SCREEN, qui permet de lire un caractère (code Ascii + attributs) présent à une position donnée de l'écran Text.
- CURSET, qui permet de modifier un numéro de couleur (0 à 255) présent à une position donnée de l'écran Hires (et bien sûr de positionner le curseur Hires).
- POINT, qui permet de lire un numéro de couleur (0 à 255) présent à une position donnée de l'écran Hires.
- Les commandes Hires permettent de "dessiner" directement dans la banque secondaire. Mais pour que ce dessin soit effectif à l'écran, il faut rafraîchir la Ram vidéo à l'aide de la commande REFRESH (RELEASE dans les versions ultérieure de la Rom Super-Oric).
- Outre les classiques commandes Hires DRAW, CIRCLE, CURMOV, Fabrice a introduit FILL pour remplir des plages délimitées par un contour (voir plus haut).
- CLOAD INK, qui permet de copier des palettes directement de la mémoire flash à la bonne place dans la banque secondaire.
- CLOAD CHAR, qui permet de copier des définitions de caractères directement de la mémoire flash à la bonne place dans la banque secondaire.
- CLOAD TEXT, qui permet de copier un écran Text directement de la mémoire flash à la bonne place dans la banque secondaire.

- CLOAD HIRES, idem avec un écran Hires.

Cette délocalisation des écrans Text et Hires dans la banque secondaire ($7F) entraîne de nombreuses conséquences: L'adressage à partir de la Rom située dans la banque principale ($7E) doit donc se faire avec un adressage long (sur 3 octets au lieu de 2). Cela touche, entre autres, les pointeurs Text et graphique, qui doivent passer sur 3 octets.

Un simple exemple : dans la page 0, l'adresse de la ligne du curseur Hires (classiquement en $10-$11) s'étend maintenant de $10 à $12. Aie! Il faut décaler l'adresse de la ligne du curseur Text (classiquement en $12-$13) pour qu'il utilise $13 à $15. Du coup, il faut mettre ailleurs les variables qui se trouvaient en $14 et $15. Fabrice a dû avoir des sueurs froides avec tous ces "dérapages"! Je vous laisse imaginer le nombre de patches qu'il a dû appliquer rien que pour résoudre ce genre de problèmes!

f) Adressages ″longs″. Comme nous venons de le voir, la Rom Basic, localisée en Banque $7E (banque dite principale), communique avec le reste de l'espace mémoire de la Snes (ou au moins certaines parties). Dans la banque $7F (banque dite secondaire), elle lit/écrit des données relatives aux définitions des caractères et des couleurs, ainsi qu'aux écrans Text et Hires. Mais elle communique aussi avec la banque $80 où se trouve le "code 65816" assurant l'initialisation de et l'interfaçage avec du hardware de la Snes. Elle appelle notamment des routines situées dans cette banque $80. Cela implique d'utiliser un adressage "long" et donc du code 65816.

Par exemple la commande REFRESH appelle la routine "hires_transfer", qui lance un transfert par DMA de l'écran Hires (situé en banque $7F) vers la VRAM (Ram vidéo) et donc vers l'écran physique de la Snes. Cette routine située en $8004 de la banque $80 a donc pour adresse "longue" $808004 et l'appeler depuis la Rom Basic nécessitera un jsr.l $808004.

Ce jsr.l est mis en place dans la Rom Basic sous la forme suivante, compréhensible par l'assembleur 6502 Frankenstein: db $22, $04, $80, $80, soit insertion pure et simple des octets $22, $04, $80 et $80. Cet exemple ouvre des perspectives grandioses: à nous l'espace faramineux de la mémoire de la Snes!

g) Changement du hardware. Les différentes routines de reset et d'initialisation et quelques commandes telles que WAIT doivent tenir compte du changement de hardware entre l'Oric classique et le Super-Oric hébergé dans la Snes. La Snes est riche de registres localisés non en page 2 ou 3, mais aux adresses $21xx et 42xx accessibles dans plusieurs banques, notamment $00 et $80, mais pas $7E! On a à nouveau besoin d'adressage long. Je n'insiste pas étant donné la complexité des relations avec le hardware de la Snes.

h) Le son (non encore implémenté dans la Rom 0.01). Enfin, la Rom Basic sera encore ultérieurement modifiée pour bénéficier des possibilités sonores de la Snes. Les commandes relatives au son et leurs paramètres seront profondément modifiés à partir de la Rom Super-Oric v1.31.

La partie "code 65816"

Rappelons que cette partie assure l'interfaçage entre le Basic 1.1 modifié et le hardware de la Snes. C'est là que ça se complique! Heureusement, dans la version 0.01, ce code n'occupe "que" #231 octets. Et c'est déjà énorme quand on ne connaît rien à la Snes, car les dessous de cette belle machine sont infernaux! Je vous recommande le Manuel de Programmation de la Snes par ″Olorin113″, que vous trouverez à l'url:

http://www.stratix.fr/base/LivreManuels/SnmpV4r01a.rar (Merci François S.). C'est LE manuel idéal pour améliorer vos maux de tête, bref... incontournable! Attention toutefois, il contient quelques erreurs, ce qui est inévitable pour un ouvrage de cette envergure.

Dans la Rom Super-Oric, le code 65816 est situé au départ de $008000 à $008230, c'est à dire dans la mémoire flash de la cartouche. Mais, c'est une caractéristique de la Snes, il a une image miroir dans la banque $80 de $808000 à $808230 et c'est là qu'il est exécuté au final. En effet la Banque $80 autorise un accès, et donc une exécution, plus rapide.

A) Remarque préliminaire.

Ce "code 65816" est exécuté en mode natif 65816, dont je ne connais pas bien les subtilités. Je me suis fait piéger par certains mnémoniques qui mettent en jeu un nombre variable d'octets selon la largeur des registres utilisés (une phrase difficile à comprendre, mais essentielle).

Lors de l'initialisation de la Snes pour le Super-Oric, l'exécution de la routine "start" met les index X et Y sur 16 bits et l'accumulateur sur 8 bits (avec les instructions REP #$10 et SEP #$20 respectivement).

Par la suite, le code de certains mnémoniques occupera 3 octets au lieu de 2. Par exemple le code $A2 (LDY#constante) prend maintenant en compte les deux octets qui suivent. Ainsi le code hexadécimal A20000 signifie LDY #$0000 et non LDY #$00 suivit de BRK, comme l'indique le désassembleur que j'ai utilisé! (65816 SNES Disassembler v2.0 by John Corey). L'assembleur Snesasm utilisé par Fabrice utilise un pseudo mnémonique ldy.w #$0 pour obtenir le bon résultat. Le ".w" indique qui faut manipuler un word (2 octets).

Autre exemple, là où le désassembleur de John Corey indique CPX #$00, suivit de RTI (code $40), il faut utiliser cpx.w #$4000 pour obtenir le code voulu, soit E00040.

Les tables recensant les mnémoniques indiquent presque toutes que le code A2, qui correspond à LDY#constante est commun au 6502 et au 65816 et occupe 2 octets. Rares sont celles qui précisent une nuance dans le cas du mode "natif" 65816: En effet, LDY#contante occupe 3 octets dans le cas où le registre Y est mis à 16 bits. Idem pour CPX #constante et pour tous les mnémoniques utilisant le mode immédiat sur les index X ou Y et sur l'accumulateur A.

Lorsque X et Y sont mis en 16 bits, cela concerne les mnémoniques CPX, CPY, LDX et LDY. Lorsque l'accumulateur A est mis en 16 bits, cela concerne les mnémoniques ADC, AND, BIT, CMP, EOR, LDA, ORA et SBC.

B) Que trouve-t-on dans ce "code 65816"?

1) Routine Hires_transfert

En $8004, un JSR $8168. A cette adresse se trouve la routine hires_transfer, qui lance le transfert DMA de l'écran Hires (en banque $7F) dans la VRAM (Ram vidéo). Cette routine est appelée par la commande REFRESH à partir de la Rom Basic.

2) Routine Start

En $8008, cette routine réalise un certain nombre d'initialisations:
Initialise les index X et Y sur 16 bits.
Initialise l'accumulateur sur 8 bits.

JSR init_ppu en $81AA
JSR clear_vram en $80A8
JSR init_charset en $8055
JSR init_graphscreen en $8036
JSR init_palette; en $80BC
Allume l'écran avec la plus forte luminosité.
Initialise le registre de l'écran (INIDISP)
Initalise le registre de banque de donnée (DBR) pour $7E (banque Oric principale)
JSR copyrom en $8061
Initialise les index X, Y et l'accumulateur sur 8 bits.
Mets l'accumulateur B à 1, afin que les TCS laissent S en page 1.
Mets les index X et Y à zéro.
Et finalement exécute un JMP cold_reset_Basic_Oric en $7EF88F
Voilà, l'essentiel est fait... enfin presque!

3) Routine init_graphscreen

En $8036, cette routine dont le rôle semble être d'initialiser l'écran Hires dans le mode BG1 (arrière plan 1). J'avoue que son contenu est du chinois pour moi.

4) Routine init_charset

En $8055, cette routine mets les caractères dynamiques à zéro. En fait j'ai l'impression que cette routine mets les 64 Ko de la banque $7F à zéro!

5) Routine copyrom

En $8061, cette routine copie la Rom Basic, située à l'origine de $8231 à $C230 dans la banque $00 (mémoire flash de la cartouche), vers la banque $7E (banque principale) de $C000 à $FFFF où elle sera exécutée.

6) Routine nmi_handler

En $8072, cette routine effectue un reset NMI interrupt (remise à zéro par simple lecture en $4210), une remise à zéro du BG 2 Horizontal Scroll Offset, un jsr dma_palette en JSR $810C (transfert DMA de la palette vers la Ram graphique), un jsr dma_screen en $8136 (transfert DMA de l'écran Texte vers la Vram), un jsr dma_charset en $80DA (transfert DMA des caractères dynamiques vers la Vram).

7) Routine irq_handler

En $8097, cette routine effectue un reset IRQ (remise à zéro par simple lecture en $4211) et simule une interruption Timer 1.

8) Routine clear_vram

En $80A8, comme son nom l'indique, cette routine remet à zéro la Vram.

9) Routine init_palette

En $80BC, cette routine met à zéro la zone des palettes en banque $7F, soit de $7FF800 à $7FF9FF, sauf $7FF802 et $7FF803 qui sont mis à #FF.

10) Routine dma_charset

En $80DA, cette routine transfère par DMA les #1000 octets définissant les caractères dynamiques situés de $7F0000 à $7F0FFF dans la banque secondaire, vers la Ram vidéo (Vram). Soit 4 octets de définition pour chacun des 256 caractères dynamiques en mode Lores 0 ou 8 octets de définition pour chacun des 128 caractères dynamiques en mode Lores 1.

11) Routine dma_palette

En $810C, cette routine transfère par DMA les #200 octets (soit 2 octets de définition pour chacune des 256 couleurs) situés de $7FF800 à $7FF9FF dans la banque secondaire, vers la Ram Graphique (CG-RAM).

12) Routine dma_screen

En $8136, cette routine transfère par DMA les #800 octets (soit 2 octets pour chacun des 32x28 caractères de l'écran Text, ce qui donne en fait #700 octets = mystère!) situés de $7FF000 à $7FF7FF dans la banque secondaire, vers la Ram vidéo (Vram).

13) Routine hires_transfer

En $8168, c'est la routine appelée en $8004. Cette routine lance le transfert par DMA des #E000 octets de l'écran Hires situé de $7F1000 à $7FEFFF dans la banque secondaire, vers la Ram vidéo (Vram). Cette routine est appelée par la commande REFRESH à partir de la Rom Basic. Soit 1 octet de définition (un numéro de couleur compris entre 0 et 255) pour chacun des 256x224=57344 pixels (#E000) de l'écran Hires.

14) Routine init_ppu

En $81AA, cette routine initialise le Picture Processing Unit. De nombreux registres du PPU (de $2100 à $21FF) sont initialisés. Leur rôle n'est pas toujours clair. Les modes BG1 et BG2 sont activés.

15) Routine init_charset_16_colors

En $8205 se trouve cette routine dont le nom évoque le mode Lores 1. Cependant, chaque caractère est défini sur 16 octets (8 words) et cela correspond à Lores 0, plutôt qu'à Lores 1, qui nécessite 32 octets (16 words) par caractère.

Cette routine copie les définitions de caractères du jeu normal de l'Oric (situés dans la Rom Basic, de $FC78 à $FF77, soit #300 octets) vers la zone des caractères dynamiques de la banque $7F. Cependant, 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.

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. Les 96 caractères normaux, définis à raison de 16 octets par caractères (soit 96x16=1536 ou #600), sont donc copiés de $7F0200 à $7F07FF.

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, la seconde (sorte d'homologue inverse) de $7F0800 à $7F09FF.

Cette dernière routine se termine en $8230 et marque la fin du "code 65816". Le fichier source Oric2.s est alors complété pour insérer la Rom Basic modifiée (basic2.rom, v0.01) à la suite de ce "code 65816". Enfin deux patches sont installés en $FFE6 et en $FFFA, c'est à dire vers la fin de la Rom Super-Oric, pour disposer un certain nombre de vecteurs dont la Snes a besoin pour fonctionner.