# Codes Secrets

*Par André C.*

Cet article est pour tous ceux qui ont quelque chose à cacher. Non, je rigole, cet article est pour tous les curieux de l'Oric. Au cours de mes pérégrinations récentes, je suis tombé sur une curiosité des plus curieuses. Il s'agissait de code machine camouflé en zone de data. Et c'est de ça que je veux vous parler.

### Etat de la question

Comme vous le savez, notre microprocesseur, le 6502, accepte en entrée des codes sur un octet, suivis éventuellement d'un ou deux octets comme argument. Par exemple, si on lui envoie la séquence #A9, #00, il comprend "Charge l'accumulateur A avec la valeur qui suit" et exécute A=#$00. En langage assembleur cela donne LDA #$00 ce qui est nettement moins abscons.

Vous pourriez penser que sur un octet, on peut coder 256 valeurs de #00 à #FF et donc que le 6502 peut comprendre 256 commandes différentes. Il n'en est rien. Les ingénieurs qui ont conçu le 6502 avaient besoin d'environ 150 commandes et ont utilisés environ 150 octets. Ils ont conçu des circuits logiques électroniques capables de réaliser ces commandes. En fait, chacun des bits qui composent un octet de commande, déclenche tel ou tel type d'action ou mode d'adressage. Je n'entre pas plus loin dans le détail.

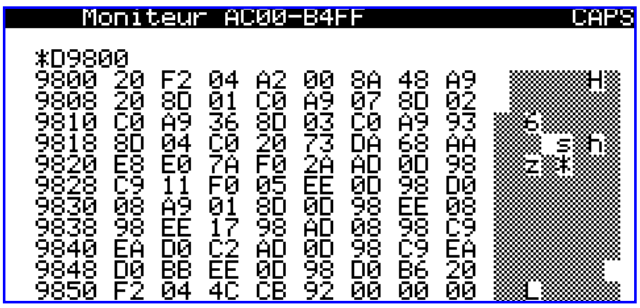
Est-ce par hasard ou par analogie avec d'autres microprocesseurs ou est-ce le fruit d'une curiosité quasi maladive, toujours est-il qu'il s'est trouvé des cinglés (à l'époque le mot "geek" n'existait pas encore) pour soumettre au 6502 la centaine d'octets qui n'avaient pas d'usage documenté. Evidemment, il fallait envoyer ces octets, un par un et en sortie examiner le résultat sur les registres A, X, Y, S (stack), PC (program counter), et P (regroupant les drapeaux N, V, B, D, I, Z, et C). Quel boulot de fou !

Si cela vous tente d'avoir plus d'information sur les "6502 extra codes", "6502 undocumented opcodes" ou "6502 Illegal opcodes", vous les trouverez facilement sur internet en entrant ces expressions dans un moteur de recherche. Une très vaste documentation existe là-dessus. Pour faire simple, voyez <http://nesdev.com/undocumented_opcodes.txt> qui indique aussi les drapeaux affectés par opcodes "non officiels".

Sachez que la plupart de ces opcodes sont des variantes ou des combinaisons des codes officiels. Parmi tout ça, il en existe qui ne font rien (comme le NOP officiel, qui ne fait strictement rien) ou qui se contentent de modifier un ou plusieurs drapeaux (comme le BIT officiel, qui n'affecte que N, V et Z). Outre BIT, les opcodes "officiels" qui tombent dans cette deuxième catégorie sont nombreux : CLC, CLD, CLV, CMP, CPX, CPY, SEC et SED. Il faut remarquer que si on ne teste pas le ou les drapeaux modifiés, tout se passe comme si ces opcodes avaient été inactifs. On est alors ramené à la première catégorie, celle de NOP. Tous les autres opcodes "officiels" affectent les registres A, X, Y, S ou PC et ne sont pas de notre propos du jour.

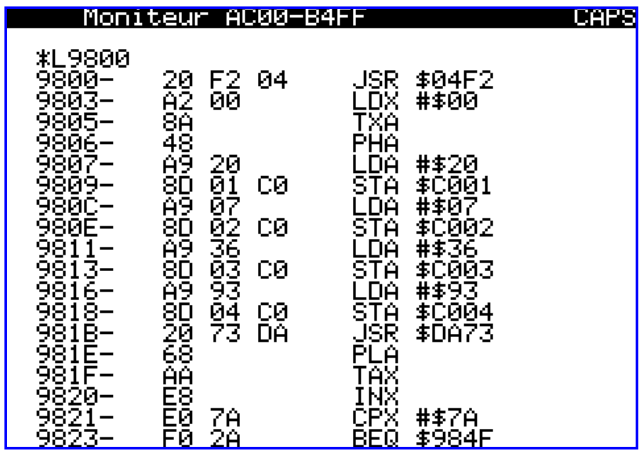
### Zone de code versus zone de data

Soit maintenant un programme langage machine localisé en mémoire à partir de l'adresse XXXX. Un simple dump de la zone mémoire correspondante ne montrera qu'une suite d'octets incompréhensibles.




Impossible de savoir s'il s'agit d'un programme ou d'une zone de data. Dans l'exemple de cette figure, il s'agit de la routine qui cherche sur la disquette les secteurs correspondant au jeu Risiko et les charge en Ram. Si on utilise un désassembleur pour lister à partir de cette adresse XXXX, la situation devient plus claire. La seconde figure (page suivante) montre le début de ce que ça donne. On voit que la zone de programme apparait comme une suite cohérente d'opcodes (suivis éventuellement de leur argument).

Par contre, les zones de data seront truffées de "??" et de commandes anarchiques. En effet, les désassembleurs ne gèrent que les opcodes offi-




ciels. Quand ils rencontrent un opcode valide, ils savent s'il y a après zéro, un ou deux octets en argument. Ils transcrivent alors en clair ce qu'ils ont trouvé. Mais s'ils rencontrent un octet dont ils ne savent que faire, ils le remplacent par "??" et essaient de se resynchroniser sur l'octet suivant, qui par hasard peut correspondre à un opcode valide (avec ou sans argument), mais cette "chance" ne dure pas et ils retombent bientôt un octet incongru. Le résultat est une suite de tentatives de désassemblage incohérente. Il est donc facile d'y reconnaître une zone de data.

**Principe du camouflage d'une zone de code**

Je n'ai trouvé aucun désassembleur dans le monde Oric, qui reconnaisse les opcodes non officiels. Pour camoufler un programme machine, il n'y a donc rien de plus simple que de le truffer d'opcodes non officiels, choisis pour ne pas perturber le déroulement du programme (un peu comme si on parsemait le programme de nombreux NOP). Le désassembleur va remplacer ces opcodes non officiels par des "??". Le résultat est une désorganisation apparente et un listing incohérent (voir plus loin l'exemple de la routine de transcodage utilisée par Daniel Duffau pour Risiko).

L'utilisation de NOPs donnerait le même résultat sur le déroulement du programme, mais le listing ne serait pas désorganisé, les NOPs étant reconnus comme tels (voir plus loin la version "corrigée" du même listing où les opcodes non officiels ont été remplacé par des NOPs).

**Opcodes "non officiels" les plus intéressants**

Il existe notamment une série de 6 faux NOP, qui sont exactement comme les NOP, mais avec un autre octet à la place du #EA normal. Ce sont : #1A, #3A, #5A, #7A, #DA et #FA, qui seront traduit par le désassembleur par des "??" alors que le 6502, lui, les avale sans broncher, comme si c'étaient des NOP !

Et il n'y a pas que les faux NOP, il y les 13 DOP, qui ne font rien et utilisent même un argument sur un octet (qui ne joue aucun rôle). Ils sont appelés DOP (double NOP, car le résultat est que deux octets sont inactifs). Les codes correspondants sont #14, #34, #44, #54, #64, #74, #80, #82, #89, #C2, #D4, #E2 et #F4. Ils sont compris par le 6502, qui néglige alors aussi l'octet suivant, comme s'il y avait deux NOP (#EA, #EA). MAIS le désassembleur, ne les connaissant pas, va mettre un "??" pour le premier des deux et éventuellement repartir sur une fausse base pour le suivant. Si par malheur l'octet suivant (qui est négligé par le 6502, mais pas par le désassembleur) est un code valide (un code correspondant à un opcode officiel). Il s'en suit une grosse désorganisation du listing !

Et ce n'est pas fini ! Vive les 6 TOP (triple NOP), analogues aux DOP, mais avec un pseudo argument sur deux octets ! Les codes correspondants sont : #1C, #3C, #5C, #7C, #DC et #FC. Eux aussi sont compris par le 6502, qui néglige alors les 2 octets suivants, comme s'il y avait trois NOP (#EA, #EA, #EA). MAIS je vous laisse imaginer le désarroi des désassembleurs, qui vont patiner un max !

Voilà donc déjà 25 codes "non officiels" complètement inoffensifs, qui peuvent rendre complètement indétectable une routine en langage machine, qu'il soit examiné avec un dump ou avec un désassembleur. Je remercie au passage Fabrice qui, avec l'exécution pas à pas du débogueur d'Euphoric, m'a permis de percer ce mystère !

Je n'aborderais pas les dizaines d'opcodes qui n'affectent que des drapeaux et restent donc sans effets tant qu'on ne teste pas ces drapeaux. Mais j'en donnerai un exemple tout de suite.

**Exemple de programme "camouflé"**

Voici l'exemple de la routine de transcodage qui rend opérationnel le jeu Risiko. La version du jeu présente sur disquette est inutilisable telle quelle (et encore faudrait-il savoir où elle se cache sur la disquette). La routine $9800 charge les secteurs appropriés en Ram de $1A36 à $92CB. Puis la routine $9301 transcode chaque octet de cette zone, donnant ainsi naissance au programme proprement dit.

Pour l'auteur de la protection, il était crucial de planquer ces routines, notamment celle de transcodage. Voici donc, cote à cote la routine camouflée par des opcodes "non officiels" et la routine rendue lisible par remplacement de ces opcodes par des NOPs (voir page suivante).

### Listing du désassembleur

```
92CD- A0 8F     LDY #$8F
92CF- 2B        ???
92D0- 24 84     BIT $84
92D2- 1B        ???
92D3- 2B        ???
92D4- 05 A2     ORA $A2
92D6- F8        SED
92D7- 2B        ???
92D8- 36 86     ROL $86,X
92DA- 1C        ???
92DB- 2B        ???
92DC- 25 A0     AND $A0
92DE- CB        ???
92DF- 2B        ???
92E0- 24 84     BIT $84
92E2- 03        ???
92E3- 2B        ???
92E4- 35 A2     AND $A2,X
92E6- 92        ???
92E7- 2B        ???
92E8- 21 86     AND ($86,X)
92EA- 04        ???
92EB- 2B        ???
92EC- 25 A0     AND $A0
92EE- 36 2B     ROL $2B,X
92F0- 24 84     BIT $84
92F2- 00        BRK
92F3- 2B        ???
92F4- 35 A2     AND $A2,X
92F6- 1A        ???
92F7- 2B        ???
92F8- 21 86     AND ($86,X)
92FA- 01 2B     ORA ($2B,X)
92FC- 26 86     ROL $86
92FE- 02        ???
92FF- 2B        ???
9300- 31 A0     AND ($A0),Y
9302- 00        BRK
9303- 2B        ???
9304- 36 B1     ROL $B1,X
9306- 00        BRK
9307- 45 02     EOR $02
9309- 91 00     STA ($00),Y
930B- 2B        ???
930C- 06 C8     ASL $C8
930E- 2B        ???
930F- 16 C4     ASL $C4,X
9311- 03        ???
9312- F0 0B     BEQ $931F
9314- 2B        ???
9315- 16 98     ASL $98,X
9317- D0 EA     BNE $9303
9319- 2B        ???
931A- 46 E6     LSR $E6
931C- 01 D0     ORA ($D0,X)
931E- E4 2B     CPX $2B
9320- 56 A6     LSR $A6,X
9322- 01 2B     ORA ($2B,X)
9324- 24 E4     BIT $E4
9326- 04        ???
9327- D0 DA     BNE $9303
9329- 2B        ???
932A- 8D 4C 36  STA $364C
932D- 1A        ???
932E- 00        BRK
```

### Même listing après corrections et avec commentaires

```
92CD- A0 8F     LDY #$8F    ; détourne le vecteur "Ready"
92CF- EA EA     NOP NOP
92D1- 84 1B     STY $1B     ; en $1B-$1C
92D3- EA EA     NOP NOP
92D5- A2 F8     LDX #$F8    ; normalement jmp $CCB0
92D7- EA EA     NOP NOP
92D9- 86 1C     STX $1C     ; vers $F88F (reset à froid)
92DB- EA EA     NOP NOP
92DD- A0 CB     LDY #$CB    ; initialise $03-$04
92DF- EA EA     NOP NOP
92E1- 84 03     STY $03     ; avec l'adr $92CB
92E3- EA EA     NOP NOP
92E5- A2 92     LDX #$92    ; fin de la zone à transcoder
92E7- EA EA     NOP NOP
92E8- 86 04     STX $04     ; initialise $00-$01
92EB- EA EA     NOP NOP
92ED- A0 36     LDY #$36    ; avec l'adr $1A36
92EF- EA EA     NOP NOP
92F1- 84 00     STY $00     ; début de zone à transcoder
92F3- EA EA     NOP NOP     ; puis adr de transcodage
92F5- A2 1A     LDX #$1A    ; et initialise $02
92F7- EA EA     NOP NOP
92F9- 86 01     STX $01     ; avec #$1A qui est le
92FB- EA EA     NOP NOP
92FD- 86 02     STX $02     ; code de transcodage
92FF- EA EA     NOP NOP
9301- A0 00     LDY #$00    ; index lecture/écriture
9303- EA EA     NOP NOP
9305- B1 00     LDA ($00),y ; lect. octet à transcoder
9307- 45 02     EOR $02     ; contient #$1A masque transc.
9309- 91 00     STA ($00),Y ; écrit. octet transcodé
930B- EA EA     NOP NOP
930D- C8        INY         ; incrémente index pour suiv.
930E- EA EA     NOP NOP
9310- C4 03     CPY $03     ; $03 contient LL adr fin
9312- F0 0B     BEQ $931F   ; branche si Y=#$CB
9314- EA EA     NOP NOP
9316- 98        TYA         ; pour positionner drapeau Z
9317- D0 E0     BNE $9303   ; reboucle page pas finie
9319- EA EA     NOP NOP
931B- E6 01     INC $01     ; 256 octets ont été traités
931D- D0 E4     BNE $9303   ; rebouclage forcé
931F- EA EA     NOP NOP
9321- A6 01     LDX $01     ; pour tester HH adr transc.
9323- EA EA     NOP NOP
9325- E4 04     CPX $04     ; sera fini pour HH=$92
9327- D0 DA     BNE $9303   ; reboucle si pas fini
9329- EA EA     NOP NOP
932B- 4C 36 1A  JMP $1A36   ; fini, lance le prg Risiko
932E- 00        BRK
```

### Conclusion

Ce type de protection est très difficile à détecter. Dans l'exemple donné, le camouflage aurait pu être pire si au lieu d'utiliser toujours le même opcode non officiel, l'auteur avait un peu varié. En effet la répétition de cet octet #2B a fini par attirer mon attention. Mais comme je l'ai déjà dit, le débogueur d'Euphoric, permet de suivre l'exécution pas à pas et de se rendre compte que l'octet #2B et l'octet suivant n'avaient aucun effet. Merci Fabrice, grâce à toi j'ai fini par comprendre. Je ne vous encourage pas à protéger vos programmes Oric contre la copie, faites en plutôt profiter la communauté. J'espère que cet article aura eu le mérite de vous divertir un peu...