Visitons la Rom Monitoring

(3e partie) par André Chéramy et Claude Sittler

P) Modification de la commande CALL en E946-E94B.

Syntaxe CALL#xxxx,A#xx,X#xx,Y#xx

Les adresses et la valeur de l'octet peuvent être données en décimal (sans le '#' bien sûr). Permet de lancer la routine dont l'adresse est indiquée soit de la manière habituelle, soit en fixant la valeur des registres A, X et Y. Mais l'utilisation de cette nouvelle commande CALL sans indiquer de valeur pour A, X et Y est dangereuse, car en absence de ces arguments, c'est l'ancienne valeur stockée respectivement en 06, 07 et 08 qui sera utilisée pour initialiser chacun de ces 3 registres. Tout à la fin de la commande le JMP (0033) a été remplacé par un JMP FE67 (nouvelle routine)

E949- 4C 67 FE JMP **FE67** initialise registres A, X et Y

Q) Anciennes Cdes STORE et RECALL en E987-EAC0.

a) Nouvelle routine «Décode une instruction et affiche la ligne correspondante». Elle est appelée par la commande DES, dont il s'agit de la routine principe. En entrée on a X=#00.

E987- 20 5B F8 JSR **F85B** va la ligne, affiche l'adresse de désassemblage en hexadécimal et affiche un espace

E98A- A1 33 LDA (33,X) récupère l'octet en 33-34, c'est l'octet à décoder

Il y a 56 mnémoniques différents, avec plusieurs modes d'adressage pour chacun, soit un réseau inextricable de combinaisons. De plus, parmi les 256 codes possibles, certains ne sont pas utilisés pour des instructions et ne sont rencontrés que dans les données, qui elles peuvent prendre toutes les valeurs possibles de #00 à #FF.

La commande DES ne peut pas savoir si l'octet dont vous indiquez l'adresse est une instruction (ce qui est généralement le cas) ou une donnée. Chaque instruction est suivie d'un nombre déterminé d'octets de données, puis vient l'instruction suivante. C'est donc seulement la place d'un octet dans le flot du code qui détermine s'il s'agit d'une instruction ou d'une donnée.

DES tentera de décoder une instruction, même si vous lui indiquer l'adresse d'une donnée. Si l'octet ne correspond au code d'aucune instruction, il affichera '???' à la place du mnémonique et tentera de décoder le suivant. Si par malheur l'octet indiqué est une donnée, mais correspond au code d'une instruction, DES tentera de décoder, mais tôt ou tard il tombera sur une incohérence, affichera '???' et tentera de se re-synchroniser en examinant le suivant.

Dans ce qui suit, nous serons amenés à décortiquer les différents bits qui composent l'octet à décoder. Pour faciliter le raisonnement, nous utiliserons la notation la plus simple possible. Comme les 8 bits d'un octet se lisent de droite à gauche, nous conviendront que l'octet à décoder est de la forme 'hgfe dcba', correspondant à la notation plus classique 'b7b6b5b4 b3b2b1b0'. Par exemple lorsqu'on effectue un LSR (décalage logique à droite) on obtiendra '0hgf edcb' et le bit 'a' passe dans C (l'indicateur Carry, c'est à dire en général le bit de retenue). Notre notation permet de repérer la position des bits d'origine dans l'octet résultant, même après plusieurs traitements logiques.

E98C- A8 TAY copie de l'octet à décoder

E98D- 4A LSR donne '0hgf edcb' et C=a

La commande DES va traiter séparément des octets pairs et impairs, ce qui va conduire à une grande simplification, car la majorité des mnémoniques (48 sur 56) ont un code pair. A titre de curiosité, c'est le cas de : ASL BCC BCS BEQ BIT BMI BNE BPL BRK BVC BVS CLC CLD CLI CLV CPX CPY DEC DEX DEY INC INX INY JMP JSR LDX LDY LSR NOP PHA PHP PLA PLP ROL ROR RTI RTS SEC SED SEI STX STY TAX TAY TSX TXA TXS TYA. Il reste seulement 8 mnémoniques dont le premier bit 'a' est à un : ADC AND CMP EOR LDA ORA SBC STA.

E98E- 90 0B BCC E99B continue en E99B si C=0 (code pair) avec A='0hgf edcb', c'est à dire un index simplifié allant de #00 à #7F qui va permettre de lire une table contenant les modes d'adressage correspondant à chaque octet à décoder.

Dans le cas où le premier bit 'a' était à un on examine maintenant le second bit 'b'. En effet, il n'existe aucune instruction dont les 2 bits 'a' et 'b' soient tous les deux à un (c'est à dire de la forme hgfe dc11).

E990- 4A LSR donne '00hg fedc' et C=b

E991- B0 17 BCS E9AA ce n'est pas un mnémonique

En effet, les octets des 8 mnémoniques ADC AND CMP EOR LDA ORA SBC STA du deuxième groupe sont de la forme initiale 'hgfe dc01'. Un examen approfondi des instructions dont le code est impair révèle qu'elles sont distribuées régulièrement : leur code hexadécimal se termine toujours par '1' '5' '9' ou 'D'. Il n'y a qu'une seule exception, l'octet #89 n'est pas une instruction. Il faut donc éliminer ce cas. C'est ce que fait ici le programme.

E993- C9 22 CMP #22 est-ce #22, ('0010 0010') ? Cette valeur actuelle correspond à '1000 10ba' au départ or nous traitons ici des codes de la forme 'hgfe dc01', donc le code cherché est '1000 1001' soit #89.

E995- F0 13 BEQ E9AA si #89, pas un mnémonique

Résumons un peu pour clarifier nos esprits. Restent donc en course, à ce niveau, les octets impairs de la forme initiale 'hgfe dc01' et qui ont la forme actuelle A='00hg fedc', c'est à dire un index simplifié allant de #00 à #3F (soit de 0 à 63 pour 64 octets à décoder). Cet index va permettre de lire une table contenant les modes d'adressage correspondant à chaque octet à décoder. Or un examen attentif de la répartition des modes d'adressages pour ces 64 octets révèle qu'ils se répètent régulièrement selon une séquence immuable de 8 modes d'adressage correspondant aux octets à décoder de valeur #01, #05, #09, #0D, #11, #15, #19, #0D, puis #21, #25...puis #41, #45...puis #61, #65... etc. Cette série se répète sur toute l'étendue #00 à #FF selon un modulo de #20, c'est à dire 8 fois (on a bien 8x8=64 octets à décoder). Il est donc possible de simplifier l'index actuel allant de #00 à #3F en un index allant de #00 à #07, c'est à dire pointant uniquement sur le premier groupe de 8 modes d'adressage. Les cas suivants seront ramenés à cette première série.

E997- 29 07 AND #07 l'accumulateur A passe de '00hg fedc' à '0000 0edc' allant de #00 à #07 et pointant donc sur les 8 premiers modes d'adressage. Il s'agit maintenant de fusionner ce 2e groupe au 1er, afin de pouvoir accéder aux deux populations dans une seule table. Pour ce faire, il suffit de mettre les informations concertant le 2e groupe à la suite de celles concernant le 1er groupe et de modifier l'index du 2e groupe pour qu'il pointe au bon endroit. Pour le 1er groupe, on a actuellement A='0hgf edcb', c'est à dire un index allant de #00 à #7F (voir en E98E). Pour le 2e groupe, on a actuellement A='0000 0edc', c'est à dire un index simplifié allant de #00 à #07. Il suffit donc d'ajouter #80 à cet index pour qu'il pointe de #80 à #87.

E999- 09 80 ORA #80 force le bit de poids le plus fort à 1 ce qui donne '1000 0edc', c'est à dire un index de #80 à #87

On aborde donc la routine suivante avec un index qui va de #00 à #7F pour la 1ère population de mnémoniques dont le code est pair et de #80 à #87 pour la 2e population de mnémoniques dont le code est impair. On pourrait utiliser cet index pour lire la 1ère table qui contient le mode d'adressage de chacune des instructions, mais il est encore possible d'optimiser. En effet, il y a 13 modes d'adressages différents, auxquels Jean-Jacques Jung a ajouté un 14ème pseudo mode pour les octets qui ne correspondent à aucune instruction. Ces modes sont codés de #0 à #D, c'est à dire sur 4 bits. On peut donc en stocker deux par octet dans cette première table, ce qui réduit la taille de la table de moitié.

Rappel : Chaque octet de 8 bits est formé de deux nibles de 4 bits et peut être noté de la manière suivante 'Nn' où 'N' représente le nible de poids fort et 'n' le nible de poids faible.

En pratique, l'index est divisé par 2 avec un LSR qui décale tous les bits vers la droite et met le bit de poids le plus faible dans l'indicateur C. On garde précieusement le bit C qui va permettre de savoir si le nouvel index désigne le nible 'N' (C=0) ou le nible 'N' (C=1).

Aucune information n'est donc perdue par cette simplification. Par pure curiosité, remarquez que C est le bit 'b' d'origine pour le 1er groupe (de code pair) et le bit 'c' d'origine pour le 2e groupe d'instructions (de code impair).

E99B- 4A LSR divise l'index par 2 et sauve le bit 'éjecté' dans C. Le nouvel index va de #00 à #43 (0 à 67) la table des modes d'adressage comportera 68 entrées. Les instructions TAX et LDA qui suivent ne modifiant pas C, c'est le dernier LSR en E99B qui déterminera la valeur de C pour le BCS en E9A0.

E99C- AA TAX index de lecture dans la table située de F816 à F859

E99D- BD 16 F8 LDA **F816**,X lit un octet dans cette table

Le code suivant sert à extraire le bon nible en fonction de C.

E9A0- B0 04 BCS E9A6 si C=1, saute les 4 LSR suiv.
E9A2- 4A LSR si C=0, effectue un décalage
E9A3- 4A LSR du nible fort 'N' (les 4 bitsde
E9A4- 4A LSR poids fort) dans le nible faible
E9A5- 4A LSR 'n' (les 4 bits de poids faible) de la valeur lue dans le tableau. On passe donc de la forme 'Nn' à la forme '0N'

E9A6- 29 0F AND #0F donne donc 0N (pour C=0) ou 0n (pour C=1)

E9A8- D0 04 BNE E9AE force la suite en E9AE, c'est à dire saute les deux instructions suivantes, correspondant au cas où l'octet à désassembler ne correspondait à aucun mnémonique (voir plus haut).

E9AA- A0 80 LDY #80 impose Y=#80 (remplace la copie de l'octet à décoder)

E9AC- A9 00 LDA #00 et A=#00 (pseudo mode d'adressage pour octets de données)

Le nible obtenu, contenant le mode d'adressage de l'octet à décoder sert d'index pour lire la 2e table en FE78-FE87.

E9AE- AA TAX index pour lire la table
E9AF- BD 78 FE LDA **FE78**,X lit un octet dans la table
E9B2- 85 50 STA 50 le stocke en 50
E9B4- 29 03 AND #03 ne garde que les 2 bits de poids faible, c'est à dire le nombre d'octets de données après l'octet d'instruction.
E9B6- 85 51 STA 51 stocke ce nombre en 51

Tout ce qui nous manque encore, ce sont les 3 lettres du mnémonique. Le code qui suit va calculer un index pour lire la troisième table contenant ces lettres. Sachant qu'il y a 56 mnémoniques différents, il faudrait utiliser une table de 168 octets. Là encore cette table a été fortement optimisée. En effet, Jean-Jacques Jung a remarqué qu'il n'avait besoin que de '?' (Ascii #3F) et des lettres majuscules de A à Y codés de #41 à #59, qu'on peut les coder sur 5 bits (de #00 à #1F) et ajouter #3F avant d'afficher. On passe alors dans la 'fourchette' de #3F à #5E, ce qui suffit largement. Il est donc possible de stocker 3 lettres de 5 bits sur deux octets. Le 16ème bit a même été aussi utilisé (voir plus loin) !

L'index de lecture est calculé par le code de E9B8 à E9D1 et c'est très compliqué. Donnons en tout de suite la structure finale, sinon ce sera incompréhensible. Partons de l'octet à décoder, de la forme 'hgfe dcba'. Il est possible de calculer un index en effectuant des calculs (opérations logiques) sur les 4 bits de poids faible 'dcba'. A l'arrivée on a :

Cas	Bits 'dcba' de l'octet à décoder	groupe de mnémoniques	index calculé
1	d 0 0 0	octets pairs	#00 à #1F
2	d 1 0 0	octets pairs	#20 à #27
3	1 0 1 0	octets pairs	#28 à #2F
4	0 0 1 0 et d 1 1 0	octets pairs	#30 à #37
5	d 0 0 1 et d 1 0 1	octets impairs	#38 à #3F
6	d c 1 1	(données)	#10

Ce tableau est organisé selon l'ordre croissant de l'index calculé. En pratique, le calcul des 6 différents cas se fait en suivant des chemins et des boucles entrelacés : un véritable labyrinthe. Avant de voir le code de E9B8 à E9D1 dans le détail, voici cas par cas le détail de la logique suivie.

Cas n°1 (mnémonique pair, de la forme 'hgfe d000') :

E9BC- A=octet à décoder 'hgfe d000' (incluant #80, pseudo code pour les octets de données). Y=3 pour 3 étapes
E9C3- LSR -> A='0hgf edcb' C=a=0 BCC E9CE
E9CE- DEC -> Y=2 non nul reboucle en E9C3
E9C3- LSR -> '00hg fedc' C=b=0 BCC E9CE
E9CE- DEC -> Y=1 non nul reboucle en E9C3
E9C3- LSR -> '000h gfed' C=c=0 BCC E9CE
E9CE- DEC -> Y=0 nul continue en E9D1
E9D1- PHA ouf, c'est fini l'index est de la forme '000h gfed' et va donc de #00 à #1F. On remarque que le pseudo code #80 pour un octet de données a l'index #10 qui correspondra bien dans la table au 'mnémonique '???'.

Cas n°2 (mnémonique pair, de la forme 'hgfe d100') :

E9BC- A=octet à décoder 'hgfe dcba' et Y=3 pour 3 étapes

E9C3- LSR -> A=‘0hgf edcb’ C=a=0 BCC E9CE
E9CE- DEC -> Y=2 non nul reboucle en E9C3
E9C3- LSR -> ‘00hg fedc’ C=b=0 BCC E9CE
E9CE- DEC -> Y=1 non nul reboucle en E9C3
E9C3- LSR -> ‘000h gfed’ C=c=1 continue en E9C6
E9C6- LSR -> ‘0000 hgfe’, LSR -> ‘0000 0hgf’, ORA #20 -> ‘0010 0hgf’ et DEC -> Y=0
E9D1- PHA ouf, c’est fini l’index est de la forme ‘0010 0hgf’ et va donc de #20 à #27.

Cas n°3 (mnémonique pair, de la forme ‘hgfe 1010’.

Ceci concerne les 6 mnémoniques : TXA (code #8A = 1000 1010), TXS (code #9A = 1001 1010), TAX (code #AA = 1010 1010), TSX (code #BA = 1011 1010), DEX (code #CA = 1100 1010), NOP (code #EA = 1110 1010). NB il n’y a aucune instruction pour #DA et pour #FA.

E9B8- A=octet à décoder ‘hgfe dcba’
E9B9- AND #8F -> A=‘h000 dcba’, TAX et Y=3
E9BF- CPX #8A -> c’est le cas pour les 8 octets de la forme ‘hgfe 1010’ continue en E9CE
E9CE- DEC -> Y=2 non nul reboucle en E9C3
E9C3- LSR -> A=‘0h00 0101’, C=a=0 BCC E9CE
E9CE- DEC -> Y=1 non nul reboucle en E9C3
E9C3- LSR -> A=‘00h0 0010’ C=b=1 continue en E9C6
E9C6- LSR -> ‘000h gfe1’, LSR -> ‘0000 hgfe’, ORA #20 -> ‘0010 hgfe’ et DEC -> Y=0
E9D1- PHA ouf, c’est fini l’index est de la forme ‘0010 hgfe’ et va donc de #20 à #2F, en fait de #28 à #2F, car le bit ‘h’ des 8 mnémoniques en question est toujours à 1. Remarque importante le cas n°3 (forme ‘hgfe 1010’) est traité AVANT le cas n°4 (forme ‘hgfe dc10’) et soustrait donc du groupe n°4 les 8 octets appartenant au groupe n°3.

Cas n°4 (mnémonique pair, de la forme ‘hgfe dc10’) :

E9BC- A=octet à décoder ‘hgfe dcba’ et Y=3 pour 3 étapes
E9C3- LSR -> A=‘0hgf edcb’ C=a=0 BCC E9CE
E9CE- DEC -> Y=2 non nul reboucle en E9C3
E9C3- LSR -> ‘00hg fedc’ C=b=1 continue en E9C6
E9C6- LSR -> ‘000h gfed’, LSR -> ‘0000 hgfe’, ORA #20 -> ‘0010 hgfe’ et DEC -> Y=1
E9C7- LSR -> ‘0001 0hgf’, ORA #20 -> ‘0011 0hgf’ et DEC -> Y=0
E9D1- PHA ouf, c’est fini l’index est de la forme ‘0011 0hgf’ et va donc de #30 à #37.

Cas n°5 (mnémonique impair, de la forme ‘hgfe dc01’) :

E9BC- A=octet à décoder ‘hgfe dcb1’ et Y=3 pour 3 étapes
E9C3- LSR -> A=‘0hgf edcb’ C=a=1 continue en E9C6
E9C6- LSR -> ‘00hg fedc’, LSR -> ‘000h gfed’, ORA #20 -> ‘001h gfed’ et DEC -> Y=2
E9C7- LSR -> ‘0001 hgfe’, ORA #20 -> ‘0011 hgfe’ et DEC -> Y=1
E9C7- LSR -> ‘0001 1hgf’, ORA #20 -> ‘0011 1hgf’ et DEC -> Y=0
E9D1- PHA ouf, c’est fini l’index est de la forme ‘0011 1hgf’ et va donc de #38 à #3F.

Cas n°6 (mnémonique impair, de la forme ‘hgfe dc11’) :

Ce cas a en fait été traité en E991 pour la forme ‘hgfe dc11’ et en E995 pour l’octet #89 (‘1000 1001’) qui lui a été rattaché, par un branchement en E9AA où la valeur de l’octet servant à calculer l’index à été forcée à #80, conduisant à un index de valeur #10, lors du calcul pour le groupe n°1 auquel #80 appartient.

Reprenons maintenant le désassemblage linéaire de la Rom.

E9B8- 98 TYA octet d’origine ‘hgfe dcba’
E9B9- 29 8F AND #8F le masque ‘1000 1111’ force
E9BB- AA TAX l’octet à ‘h000 dcba’puis le résultat passe dans X (soit un nombre de #80 à #8F)
E9BC- 98 TYA octet d’origine ‘hgfe dcba’
E9BD- A0 03 LDY #03 compteur pour 3 opérations
E9BF- E0 8A CPX #8A est-ce #8A ? (‘1000 1010’)
E9C1- F0 0B BEQ E9CE si oui, continue en E9CE avec A=‘hgfe dcba’ et Y=3
E9C3- 4A LSR décalage à droite et C = bit de poids le plus faible
E9C4- 90 08 BCC E9CE si C=0, continue en E9CE
E9C6- 4A LSR si C=1, décalage à droite et C = bit de poids le plus faible
E9C7- 4A LSR encore décalage à droite et C = bit de poids le plus faible puis le masque
E9C8- 09 20 ORA #20 0010 0000 force le 6e bit à 1
E9CA- 88 DEY décrémente le compteur
E9CB- D0 FA BNE E9C7 si Y pas nul reboucle.
E9CD- C8 INY re-positionne Y
E9CE- 88 DEY décrémente le compteur. Dans tous les cas on sort avec Y=0
E9CF- D0 F2 BNE E9C3 reboucle si Y n’est pas nul
E9D1- 48 PHA c’est fini, empile le résultat

Maintenant, affiche l’octet à décoder, suivi de ou 2 octets de données (ou des espaces équivalents) et enfin de 2 espaces pour bien séparer du champ de désassemblage qui va suivre.

E9D2- B1 33 LDA (33),Y lit octet à décoder (Y=0) ou données correspondantes (Y>0)
E9D4- 20 34 FF JSR **FF34** affiche en hexadécimal
E9D7- C4 51 CPY 51 positionne C=0 si Y < octet présent en 51 (nombre d’octets de données suivants l’octet de l’instruction). Exemple : Au premier passage Y=0 donc C est positionné à 0 s’il ya 1 ou 2 octets de données à afficher et C=1 s’il n’y a pas de données.
E9D9- C8 INY pointe prochain octet à lire
E9DA- 90 F6 BCC E9D2 reboucle s’il reste des données à afficher
E9DC- A2 02 LDX #02 pour les 2 espaces
E9DE- 20 16 CD JSR **CD16** affiche X espaces

Il faut encore afficher des espaces correspondant aux éventuelles données manquantes.

E9E1- C8 INY au premier passage ici Y=nombre d’octets de données +2
E9E2- A2 02 LDX #02 2 espaces par octet manquant
E9E4- C0 04 CPY #04 positionne C=0 si Y < 4 Par exemple, au premier passage ici, s’il n’y avait pas d’octet de données, Y serait à 2, ce qui est < 4, donc C serait positionné à zéro. Il faudra 2 passages supplémentaires pour afficher les 4 espaces manquants. S’il n’y avait qu’un octet de données, Y serait à 3, ce qui est < 4, donc C serait encore positionné à zéro, autorisant une boucle supplémentaire pour afficher 2 espaces. Enfin s’il y avait 2 octets de données, Y serait à 4, ce qui n’est pas inférieur à 4, donc C serait positionné à 1 interdisant l’affichage d’espaces supplémentaires.
E9E6- 90 F6 BCC E9DE reboucle si C=0, sinon continue avec X=2

Il s’agit maintenant d’afficher les 3 lettres du mnémonique. Nous avons vu que les informations concernant ces 3 lettres sont stockées sur 2 octets. Plutôt que de les mettre à la suite

dans un même tableau et d'incrémenter l'index, Jean-Jacques Jung a choisit d'utiliser 2 tableaux et d'utiliser le même index.

E9E8- 68 PLA récupère le résultat de la
E9E9- A8 TAY grosse compilation pour l'utiliser comme index de lecture
E9EA- B9 92 FE LDA **FE92**,Y lit un octet dans le tableau
E9ED- 85 52 STA 52 le stocke en 52
E9EF- B9 D2 FE LDA **FED2**,Y lit un octet dans le tableau
E9F2- 85 53 STA 53 le stocke en 53

Les deux octets en 52 et 53 contiennent ensemble 3x5 bits codant les 3 caractères Ascii du mnémonique. La boucle suivante effectue 5 décalages à gauche sur 3 octets avec report du bit de débordement dans l'accumulateur : c<-b7 [A] b0<-C<-b7 [52] b0<-C<-b7 [53] b0<-0

E9F4- A9 00 LDA #00 initialise A = 0000 0000
E9F6- A0 05 LDY #05 pour 5 opérations
E9F8- 06 53 ASL 53 décalage à gauche du contenu de 53 avec b7->C et 0->b0
E9FA- 26 52 ROL 52 décalage à gauche du contenu de 52 avec C->b0 et b7->C
E9FC- 2A ROL décalage à gauche du contenu de A avec C->b0 et b7->C
E9FD- 88 DEY décompte les 5 tours
E9FE- D0 F8 BNE E9F8 reboucle pour 5 tours, ce qui introduit 5 bits dans A (partie basse d'un code Ascii). En sortie on a C=0 et 0 < A < #1F. Si on ajoute #3F à l'accumulateur, on obtient un octet de #3F à #5E correspondant au code Ascii à afficher.
EA00- 69 3F ADC #3F ajoute #3F au contenu de A pour obtenir un code Ascii
EA02- 20 D9 CC JSR CCD9 affiche le caractère Ascii
EA05- CA DEX X était resté à 2 depuis le BCC en E9E6
EA06- 10 EC BPL E9F4 recommence pour afficher 2 autres caractères
EA08- 20 D4 CC JSR CCD4 puis affiche un espace

Un dernier effort, maintenant il s'agit d'afficher le dernier champ. Ceci se fait en 3 temps :

1) Affiche de 0 à 2 caractères devant la valeur 'immédiate' ou l'adresse de l'action.

2) Affiche 1 ou 2 octets de données (valeur 'immédiate' ou adresse de l'action)

3) Affiche de 0 à 3 caractères après la valeur 'immédiate' ou l'adresse de l'action.

Les caractères affichés aux points 1 et 3 ci-dessus dépendent du mode d'adressage et sont indiqués en clair dans les tables jumelles en FE86 et FE8C dont la lecture se fait à l'aide d'un index X évoluant de 6 à 1. Ce sont les bits de l'octet stockés en 50 qui indiquent selon le mode d'adressage, s'il faut afficher un caractère ou passer le tour. Le cas particulier où il faut seulement afficher 'A' (mode 'Accumulateur') est traité à part.

EA0B- A2 06 LDX #06 index pour LDA en EA2F pour 6 opérations
EA0D- E0 03 CPX #03 est-ce que X=3 ?
EA0F- D0 1A BNE EA2B si non, saute la suite

La routine suivante (EA11-EA29) sera toujours sautée sauf lorsque X=3. Elle sert à insérer l'affichage de la valeur 'immédiate' ou de l'adresse de l'action ou de la lettre 'A'.

EA11- A4 51 LDY 51 0, 1 ou 2 octets de données
EA13- D0 08 BNE EA1D suite en EA1D si pas nul pour afficher donnée ou adresse
EA15- 06 52 ASL 52 si nul, décalage à gauche de l'octet contenu de 52 pour récupérer dans C le '16e' et dernier bit de l'ensemble 52-53 (lettres du mnémonique).
EA17- 90 12 BCC EA2B branche si ce '16e' bit était nul (c'est pas le mode Accumulateur)
EA19- A9 41 LDA #41 sinon charge la lettre ' A ' pour 'Accumulateur'
EA1B- D0 24 BNE EA41 suite pour affichage et fin
EA1D- A5 50 LDA 50 lit l'octet en 50 (ayant déjà subit 3 ASL)
EA1F- C9 E8 CMP #E8 positionne C=1 si [50] > ou = #E8 (soit '1110 1000' c'est le cas après 3 ASL si la valeur d'origine était #9D soit '1001 1101' pour mode d'adressage relatif.
EA21- B1 33 LDA (33),Y lit octet au pointeur (Y provient de l'octet en 51)
EA23- B0 21 BCS **EA46** s'il s'agit du mode d'affichage relatif, branche en EA46 pour le calcul de l'adresse relative = adresse de l'octet de l'instruction de branchement pointée en 33-34 + 1(instruction suivante) + A (offset relatif = octet de donnée) puis affichage en hexa de cette adresse.
EA25- 20 34 FF JSR **FF34** sinon, affiche en hexadécimal l'octet de donnée
EA28- 88 DEY décrémente index
EA29- D0 F2 BNE EA1D reboucle s'il y a un 2e octet de données à afficher, sinon reprend l'affichage des caractères Ascii.
EA2B- 06 50 ASL 50 C reçoit le bit de poids le plus fort
EA2D- 90 0E BCC EA3D si C=0 (rien à afficher)
EA2F- BD 85 FE LDA **FE85**,X sinon, lit Ascii à afficher
EA32- 20 D9 CC JSR CCD9 affiche le caractère Ascii
EA35- BD 8B FE LDA **FE8B**,X et lit un deuxième Ascii
EA38- F0 03 BEQ EA3D on n'affiche pas si c'est zéro
EA3A- 20 D9 CC JSR CCD9 affiche le caractère Ascii
EA3D- CA DEX décrémente index de lecture
EA3E- D0 CD BNE EA0D reboucle tant que pas nul
EA40- 60 RTS sinon sort
EA41- 4C D9 CC JMP CCD9 affiche le caractère Ascii
EA44- 60 RTS et retourne
EA45- 60 RTS ce RTS semble inutilisé

b) Nouvelle routine «Affiche l'adresse résultant de AY = adresse pointée en 33-34 + A + 1". A l'entrée, l'accumulateur A et la retenue C doivent être correctement initialisés. Elle est appelée par la commande DES.

EA46- 20 55 EA JSR **EA55** AY = adr pointée en 33-34 + A
EA49- AA TAX copie l'octet de poids faible
EA4A- E8 INX et l'incrémente
EA4B- D0 01 BNE EA4E saute l'instruction suivante si pas de retenue
EA4D- C8 INY sinon incrémente aussi HH
EA4E- 8A TXA copie le LL du résultat dans A
EA4F- 4C 2E FF JMP **FF2E** affiche en hexa le nombre AY

c) Nouvelle routine «Ajoute le nombre d'octets de données à l'adresse pointée en 33-34". Elle est appelée par la commande DES et calcule l'adresse du prochain octet à décoder. Retourne le résultat dans AY.

EA52- A5 51 LDA 51 0, 1 ou 2 octets de données
EA54- 38 SEC C=1, prépare une addition

d) Nouvelle routine «Ajoute l'accumulateur A à

l'adresse pointée en 33-34". Il s'agit en fait d'une entrée secondaire de la routine précédente, pour laquelle A et C doivent avoir été correctement initialisés. Retourne le résultat dans AY.

EA55- A4 34 LDY 34 charge l'octet de poids fort HH de l'adresse dans Y
EA57- AA TAX copie l'offset A dans X pour tester s'il est négatif, ce qui peut être le cas lors du calcul d'une adresse relative pour une instruction de branchement.
EA58- 10 01 BPL EA5B si A nul ou positif, saute l'instruction suivante
EA5A- 88 DEY si l'offset est négatif, décrémente le HH de l'adresse
EA5B- 65 33 ADC 33 A = A + 33 (l'octet de poids faible LL de l'adresse)
EA5D- 90 01 BCC EA60 termine si pas de retenue à propager dans HH
EA5F- C8 INY il y avait une retenue, incrémente HH
EA60- 60 RTS retourne

e) Entrée de la nouvelle commande DES.

Syntaxe : DES#xxxx (adresse hexadécimale) ou DESxxxx (adresse décimale). Cette commande désassemble à partir de l'adresse xxxx indiquée, ce qui semble simple. Pour chaque instruction, il suffit de d'afficher une ligne comportant :

1) L'adresse de l'instruction à décoder (4 digits sans #, puis un espace).

2) L'octet de cette instruction suivi éventuellement d'un ou deux octets de données (6 digits sans #, puis 2 espaces pour bien séparer les champs information d'origine et interprétation).

3) Les 3 lettres du mnémonique, puis un espace.

4) Les données correspondantes selon la syntaxe du mode d'adressage (# $ (adr) ,X ,Y etc.).

Mais en fait, c'est très compliqué si on ne veut pas y consacrer une place considérable. En effet, on pourrait utiliser une table à 256 entrées, de #00 à #FF, contenant le descriptif de chaque instruction (les 3 lettres du mnémonique, le nombre d'octets de données, le mode d'adressage et sa syntaxe). Il suffirait alors d'une routine capable de lire cette table en utilisant le code de l'instruction comme index et d'afficher.

Tous les programmeurs qui se sont attelés à cette tâche et ont rivalisé d'intelligence pour trouver des astuces afin de gagner de la place et donc de la vitesse d'exécution. Voici donc la solution proposée par Jean-Jacques Jung.

EA61- 20 53 E8 JSR E853 évalue l'adresse de l'octet à décoder et la met en 33-34
EA64- 20 F0 CB JSR CBF0 va à la ligne
EA67- 85 60 STA 60 résidu de mise au point
EA69- A2 00 LDX #00 initialise un index de lecture
EA6B- 20 87 E9 JSR **E987** routine principale de DES qui décode et affiche une ligne
EA6E- 20 52 EA JSR **EA52** ajoute le nombre d'octets de données à l'adresse pointée en 33-34 (adresse de l'octet qui vient d'être décodé) et retourne le résultat dans AY
EA71- 85 33 STA 33 met à jour dans 33-34
EA73- 84 34 STY 34 l'adresse de l'octet à décoder de l'instruction suivante
EA75- 20 0A E9 JSR **E90A** teste si 'espace' ou 'ESC' et gère pause ou fin
EA78- D0 EF BNE EA69 et reboucle pour décoder l'instruction suivante

f) Nouvelle routine «Demande 'T+adr' (copie adresse en 00-01) ', A+adr' (copie adresse en 02-03) et ',E+adr' (copie adresse en 33-34)».

Cette routine est appelée par la commande MOVE

EA7A- 20 E8 00 JSR 00E8 relit l'octet à TXTPTR
EA7D- C9 54 CMP #54 est-ce la lettre «T» ?
EA7F- D0 30 BNE EAB1 si non, SYNTAX ERROR
EA81- 20 E2 00 JSR 00E2 si oui, lit l'octet suivant
EA84- 20 53 E8 JSR E853 évalue nombre sur 2 octets
EA87- A5 33 LDA 33 lit le résultat en 33-34
EA89- 85 00 STA 00 et copie ce résultat
EA8B- A5 34 LDA 34 (adresse cible)
EA8D- 85 01 STA 01 en 00-01
EA8F- 20 65 D0 JSR D065 demande «,» et lit suivant

g) Nouvelle routine «Demande 'A+adr' (copie adresse en 02-03) et ',E+adr' (copie adresse en 33-34)». Appelé par SEARCH, CLEAN et MOVE, cette routine lit une adresse de début à TXTPTR et la place en 02-03 puis une adresse de fin et la place en 33-34. Il s'agit en fait une entrée secondaire de la routine précédente.

EA92- C9 41 CMP #41 est-ce la lettre «A»?
EA94- D0 1B BNE EAB1 si non, SYNTAX ERROR
EA96- 20 E2 00 JSR 00E2 si oui, lit octet suivant
EA99- 20 53 E8 JSR E853 évalue nombre sur 2 octets
EA9C- A5 33 LDA 33 lit le résultat en 33-34
EA9E- 85 02 STA 02 et copie ce résultat
EAA0- A5 34 LDA 34 (adresse de début de bloc)
EAA2- 85 03 STA 03 en 02-03
EAA4- 20 65 D0 JSR D065 demande «,» et lit suivant
EAA7- C9 45 CMP #45 est-ce la lettre «E»?
EAA9- D0 06 BNE EAB1 si non, SYNTAX ERROR
EAAB- 20 E2 00 JSR 00E2 si oui, lit octet suivant, puis
EAAE- 4C 53 E8 JMP E853 évalue un nombre sur 2 octets le résultat reste en 33-34 (adresse de fin de bloc)
EAB1- 4C 70 D0 JMP D070 SYNTAX ERROR

h) Nouvelle routine «Copie un octet pointé par 02-03 à l'adresse pointée par 00-01, incrémente les deux adresses et teste si la fin du bloc pointé en 33-34 est atteinte, retourne avec C=0 si pas fini». Cette routine est appelée par la commande MOVE (avec en entrée Y=#00).

EAB4- B1 02 LDA (02),Y lit l'octet pointé en 02-03 (adresse source)
EAB6- 91 00 STA (00),Y le copie à l'adresse pointée en 00-01 (adresse cible)
EAB8- E6 00 INC 00 incrémente LL de l'adresse en 00
EABA- D0 02 BNE EABE saute si LL pas nul
EABC- E6 01 INC 01 sinon reporte une retenue sur HH en 01
EABE- 4C 84 E5 JMP **E584** incrémente aussi l'adresse en 02-03, teste si la fin de la zone pointée par l'adresse en 33-34 est atteinte, retourne à l'appelant de EAB4 avec C=0 si pas fini.

à suivre...