Un logiciel de protection de fichiers "PROTECTOR"

UTILITAIRE : Protector

de Philippe GUIOCHON

Avec ce logiciel, protégez vos programmes ou vos données de toutes formes de « piratage », y compris celles de « spécialistes » de la copie.

Langages : langage machine 6502 ou Z 80.

Ordinateurs : Oric 1 ou machine dotée d'un Z 80.

Dans les années 1970, I.B.M. a présenté un système de codage, baptisé « Data Encryplion Standard », destiné à protéger certaines données informatisées des « regards indiscrets ». Ce procédé n'interdit en rien la recopie d'une disquette ou d'une cassette, mais il assure l'illisibilité quasi absolue des données si l'on ignore le ou les mots-clés ayant donné lieu à un cryptage. Ainsi, un mot de 8 lettres, encodé de la sorte peut présenter 256^8 valeurs différentes, ou encore $1 . 84 \cdot 10^{19}$!

Le système d'encodage repose sur un principe simple, suivant l'opération « OU EXCLUSIF », dont le tableau de fonctionnement est représenté **figure 1.**

Par exemple, si nous désirons coder le mot « DAVE » avec la clé « SOFT », nous obtiendrons une suite de quatre octets méconnaissables (**figure 2**).

L'intérêt de cette opération réside dans sa réversibilité : en effet, si nous appliquons une nouvelle fois au résultat du cryptage un « OU EXCLUSIF » avec la clé, la valeur d'origine nous est rendue (**fig. 3**).

Implantation d'un « PROTECTOR » sur Oric 1

La **figure 4** propose un modèle de carte mémoire matérialisant l'implantation du logiciel sur le micro-ordinateur Oric 1 (équipé d'un microprocesseur 6502). Pour adapter ce programme à une autre machine, il est nécessaire de modifier les adresses utilisées dans le code machine. La **figure 5** représente l'ordinogramme général de la routine, et la **figure 6** propose les instructions en langage machine nécessaires à sa programmation.

Pour stocker cette routine, il est possible d'utiliser un assembleur, mais nous vous proposons également un programme Basic (**fig. 7**).

L'opération OU EXCLUSIF est représentée par ⊕

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

Fig. 1. – Tableau de vérité de l'opération « OU EXCLUSIF ».

Les codes ASCII hexadécimaux, les mots « DAVE » et « SOFT » sont matérialisés par 44, 41, 56, 45 et 53, 4F, 46, 54.
L'opération OU EXCLUSIF entre ces deux chaînes de caractères est mise en évidence ci-dessous, en binaire, afin d'en clarifier le mécanisme :

« DAVE »	01000100	01000001	01010110	01000101
« SOFT »	01010011	01001111	01000110	01010100
Résultat	00010111	00001110	00010000	00010001

Fig. 2. – Cryptage du mot « DAVE » avec la clé « SOFT ».

Cryptage « SOFT »	00010111 01010011	00001110 01001111	00010000 01000110	00010001 01010100
Résultat	01000100 D	01000001 A	01010110 V	01000101 E

Fig. 3. – Si, au résultat du cryptage on applique la clé par une opération OU EXCLUSIF, la valeur initiale est à nouveau obtenue.

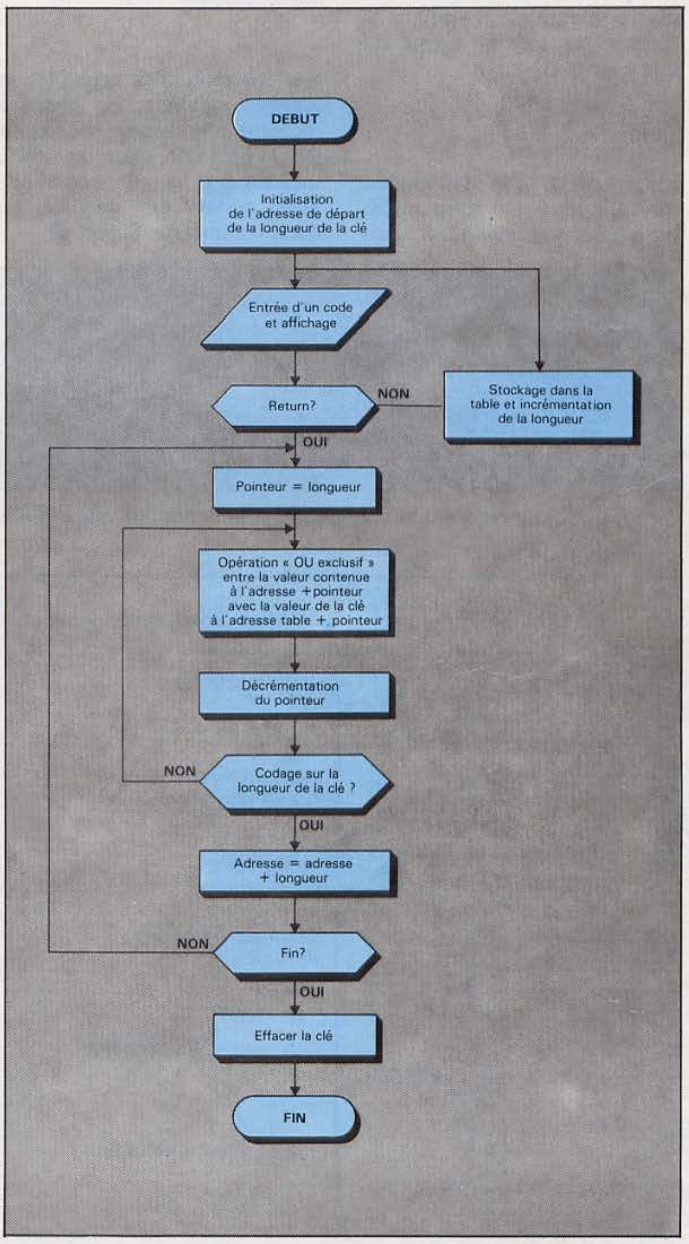

Fig. 4. – Exemple d'implantation mémoire du programme de protection.

L'utilisation de ce programme de codage est très simple : après avoir stocké « PROTECTOR » à l'adresse # B400, il vous faudra rentrer le programme ou les données à crypter.

Un CALL # B400 provoque le démarrage de la routine. Il faut alors entrer les caractères de la clé. Après avoir tapé la touche retour chariot, le codage proprement dit commence (quelques secondes pour plus de 40 K-octets). Pour constater le résultat, essayez de faire un LIST lorsque vous reprendrez la main. Dès lors, il vous sera possible de sauvegarder le programme crypté sur cassette.

Lorsqu'on désire restaurer un programme encodé (n'oublions pas qu'un tel programme n'est plus exécutable), il suffit d'effectuer les mêmes opérations que pour le cryptage :

CALL # B400
Entrée de la clé
Retour chariot

Ce logiciel peut parfaitement être adapté à un autre microprocesseur que le 6502, à condition toutefois d'en connaître le langage machine. La **figure 8** récapitule les mnémoniques de PROTECTOR pour un Z 80, sachant que les adresses nécessaires doivent être adaptées au micro-ordinateur utilisé. ■

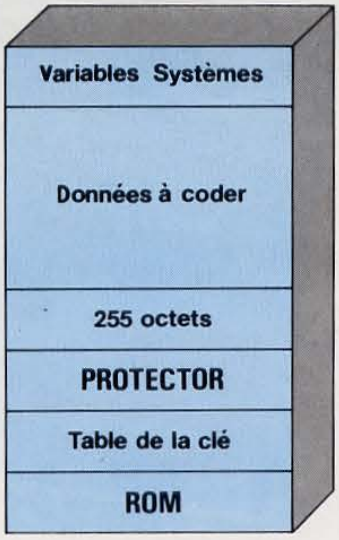

Fig. 5. – Ordinogramme de la routine de cryptage.

```
1 FOR I = #B400 TO #B43B: READ A$: A = VAL («#»+A$:
  POKE J,A: NEXT
2 DATA A9, 05, 85, 01, A2, 00, 86, 00
3 DATA 20, F8, C5, 20, 12, CC, C9, OD, FO, O7
4 DATA 9D, 01, B8, E8, 4C, 08, B4
5 DATA 8A, A8, B1, 00, 59, 00, B8, 91, 00, 88, DO, F6
6 DATA 8A, 18, 65, 00, 85, 00, A9, 00, 65, 01, 85, 01, C9, B3
7 DATA DO, E4, 9D, 00, B8, CA, DO, FA, 60
```

Fig. 7. – Programme Basic de chargement de la routine langage machine 6502.

```
        ORG   #B400     Début de l'implantation pour
                        l'Oric 1
ADR     EQU   #00       Adresse en page 0
GET     EQU   #C5F8     Adresse en ROM de la routine qui
                        charge dans l'accumulateur le
                        code de la touche enfoncée
AFFI    EQU   #EC12     Adresse en ROM de la routine qui
                        affiche le caractère de code
                        contenu dans A
TABLE   EQU   #B800     Adresse de début de la clé
DEPART  EQU   #05       0500 = adresse de début de la
                        zone à crypter
FIN     EQU   #B3       B300 = adresse de fin de la zone à
                        crypter
CR      EQU   # 0D      Code de la touche « Retour cha-
                        riot »
        LDA   ©DEPART   Initialisation de l'adresse de dé-
        STA   ADR + 1   part et de la longueur dans le re-
        LDX   @ADR      gistre X
        STA   ADR
CLAV    JSR   GET       Saisie d'un caractère et affichage
        JSR   AFFI
        CMP   @CR       Si ce caractère est le retour cha-
        BEQ   CODAGE    riot, codage
        STA   TABLE+1,X Sinon, stocker le code du carac-
        INX             tère dans la zone réservée à la clé
        JMP   CLAV
CODAGE  TXA             Y prend la valeur de la longueur
        TAY             de la clé
CLE     LDA   (ADR),Y   Exécution du OU EXCLUSIF
        EOR   TABLE,Y
        STA   (ADR),Y
        DEY             Parcours de la clé complète
        BNE   CLE
        TXA             Quand toute la clé a été traitée,
        CLC             incrément de l'adresse de codage
        ADC   ADR       avec la longueur de la clé
        STA   ADR
        LDA   @#00
        ADC   ADR+1
        STA   ADR+1
        CMP   @FIN      Tant que la fin n'est pas atteinte,
        BNE   CODAGE    on continue le cryptage
EFFA    STA   TABLE,X   Effacement de la clé et retour
        DEX
        BNE   EFFA
        RTS
```

Fig. 6. – Liste des mnémoniques 6502 du programme en langage machine.

```
        ld de, DEPART   initialiser départ, table, longueur
        ld hl, TABLE
        ld c, 0
CLAV :  call GET        saisie et affichage
        call AFFI
        cp 0D           si « return », coder
        jr z CODAGE
        ld (h1),a       sinon, stocker, incrémenter
        inc hl          la longueur, et continuer
        inc C
        JR CLAV
CODAGE : ld hl, TABLE   initialisation
        ld b, c
CLÉ     ld a, (de)      Coder sur la largeur
        xor (hl)        de la clé
        ld (de), a
        inc hl
        inc de
        djnz CLÉ
        ld a, d         si adresse < fin, continuer
        cp FIN
        jrnz CODAGE
        l hl, TABLE     sinon, effacer la clé et fin
EFF A : ld (hl), a
        inc hl
        dec C
        jrnz EFFA
        ret
```

Fig. 8. - Liste des mmémoniques Z 80 de PROTECTOR.

MINIAS.BAS

Programmation en langage machine

Après avoir présenté, dans divers numéros, le microprocesseur 6502 de Rockwell, nous vous proposons aujourd'hui ce mini-assembleur. Sans égaler les logiciels de même type, mais effectuant les traitements en plusieurs « passes », il offrira cependant aux amateurs la possibilité de programmer en langage machine et de s'initier aux charmes de la manipulation du cœur de leur micro-ordinateur. Conçu pour fonctionner sur Oric 1, il peut toutefois être aisément adapté à toutes les machines équipées d'un R 6502.

UTILITAIRE :
Un mini-assembleur 6502
de Philippe GUIOCHON

Avec ce petit interpréteur, créez vos propres routines en langage machine, vos jeux, voire des logiciels à caractère système.

Langage : Basic

Ordinateur : Oric 1

Trois avantages font de la programmation en langage machine un « must » pour optimiser l'utilisation de son micro-ordinateur : la vitesse d'exécution est sans commune mesure avec celle des programmes écrits en langage Basic ; l'encombrement des logiciels devient minime (l'espace mémoire constitue souvent une ressource très limitée sur les micro-ordinateurs familiaux - voir **figure 1**).

Enfin, le langage machine offre la possibilité de créer de nouvelles fonctions au Basic, complètement irréalisables avec un langage évolué.

L'accès à l'Assembleur n'est pas réservé à une élite « professionnelle ». L'auteur de ce programme en est la preuve. Si vous maîtrisez un tant soit peu le Basic et si vous avez quelques notions de structuration de programme, n'hésitez plus !

Le pas franchi, un problème se pose toutefois : lorsque les instructions du microprocesseur sont bien connues, comment écrire les programmes ? Les assembler à la main, puis entrer les codes en mémoire par des « POKE » est une opération très lente. Par contre, un programme spécialisé nommé « Assembleur » peut accomplir cette tâche, ne laissant à l'utilisateur que la charge de l'écriture des instructions de ses routines.

Le logiciel proposé ici est en fait un mini-assembleur.

Bien qu'il effectue le travail de conversion du langage en code machine, il ne sait cependant pas résoudre les déplacements, et il faut donc lui fournir toutes les valeurs d'adresse de manière explicite.

Le programme

Grâce à l'architecture très simple du 6502 et à ses treize modes d'adressage, il a été aisé de structurer le programme.

Les 151 instructions du 6502 sont stockées dans un tableau sous la forme « MMMCC », où MMM est la mnémonique, et CC le code hexadécimal correspondant. Afin de faciliter l'assemblage, les instructions sont coordonnées par mode d'adressage : ainsi, les 25 premiers éléments du tableau représentent l'**adressage implicite,** les 25 suivants caractérisent l'**adressage absolu,** etc.

Le programme analyse l'instruction, détermine son mode d'adressage, recherche dans la zone correspondante du tableau le code équivalent à la mnémonique et « assemble » ce code, suivi, s'il y a lieu, de l'opérande, avant d'attendre de nouveaux ordres.

Par contre, pour faciliter la programmation de cet Assembleur, nous avons dû recourir à une syntaxe extrêmement rigide (ce qui semble un peu contraignant mais améliore toujours la relecture du pro-

Les variables

- *R* : *compteur*
- *M(N)* : *contient une mnémonique et son code hexa : ex. BRK00*
- *D* : *adresse d'implantation*
- *R$* : *instruction à assembler*
- *T$* : *idem, puis simplement opérande*
- *F$* : *dernier caractère de T$*
- *L* : *longueur de T$*
- *M$* : *mnémonique de l'instruction*
- *A,B* : *pointeurs du tableau (entre A = 0 et B = 24, les modes implicites, etc.)*
- *K* : *valeur à « POKEr » ou à « DOKEr »*

```
LDA @#D1     10 A = 209
STA #65      20 POKE 101, A
LDA @#BB     30 A = 187
STA #66      40 POKE 102, A
LDX @#19     50 X = 25 : '25 lignes
LDA @#7E     60 A = 126 : 'code correspondant à un pavé gris
LDY @#24     70 Y = 36 : '36 colonnes
STA (#65).Y  80 POKE (DEEK (101)) + Y,A
DEY          90 Y = Y - 1
BNE #FB      100 IF Y <> 0 THEN 80
CLC          110 C = 0
LDA @#28     120 A = 40
ADC #65      130 A = A + PEEK (101) : IF A > = 256
                 THEN C = 1 : A = A - 256
STA #65      140 POKE 101, A
LDA @#00     170 A = 0
ADC #66      180 A = A + PEEK (102) : IF A > = 256
                 THEN C = 1 : A = A - 256
STA #66      190 POKE 102, A
DEX          200 X = X - 1
BNE #E4      210 IF X <> 0 THEN 60
RTS          220 END
```

Fig. 1. – Exemple de deux programmes assurant la même fonction (griser l'écran de l'Oric 1), l'un en Basic, l'autre en langage machine. Outre le gain de place (évident ici), le gain en vitesse d'exécution est énorme.

Mode	Exemple	Remarque
Implicite	NOP	3 caractères
Immédiat	LDA @#NN	8 caractères
Accumulateur	ASL A	5 caractères
Page zéro	LDA #NN	7 caractères
Page zéro, X	LDA #NN.X	9 caractères
Page zéro, Y	LDX #NN.Y	9 caractères
Absolu	LDA #NNNN	9 caractères
Absolu, X	LDA #NNNN.X	11 caractères
Absolu, Y	LDA #NNNN.Y	11 caractères
Relatif	BNE #NNNN	9 caractères
(Indirect, X)	LDA (#NN.X)	11 caractères
(Indirect), Y	LDA (#NN).Y	11 caractères
(Indirect)	JMP (#NNNN)	11 caractères

Tableau 1. – Liste des treize modes d'adressages disponibles sur le 6502 avec la syntaxe de programmation. Notons que le « . » est utilisé ici comme séparateur à la place de la virgule plus classique, pour des raisons de simplicité du mini-assembleur.

gramme source !). Le **tableau 1** regroupe les règles de syntaxe utilisées dans ce langage. L'espace figurant dans le tableau entre les mnémoniques et les opérandes est nécessaire. Chaque caractère des valeurs numériques, y compris le zéro, est requis. Ainsi, #03 ne doit pas être écrit # 3. Enfin, lors de l'utilisation des instructions de branchements relatifs, il suffit de prendre comme opérande l'adresse de branchement, le calcul du déplacement étant effectué par l'Assembleur.

Lorsque le mini-assembleur est lancé (par l'instruction RUN), il faut entrer l'adresse de début d'assemblage sous la forme d'une valeur hexadécimale (par exemple, #400 pour l'Oric 1).

Un point d'interrogation vous invite alors à entrer la première instruction :
#400 ? LDA @#01

Pour mémoire, le programme affichera les codes hexadécimaux correspondants, soit ici :
#400 – A9 01 LDA @#01

Toutes les valeurs utilisées ou affichées sont en hexadécimal.

Lorsque le programme est terminé, il suffit d'entrer la commande END, et votre création pourra être testée (après sauvegarde toutefois...).

La directive ORG permet de continuer l'assemblage à une autre adresse (ce qui présente l'avantage de créer des routines en une seule fois à divers endroits de la mémoire). ■

```
1 REM*********************************
2 REM
3 REM     MINI-ASSEMBLEUR 6502A
4 REM
5 REM        PHILIPPE GUIOCHON
6 REM
7 REM*********************************
9 DIMM$(150):GOTO5000
10 REPEAT
20 A=A+1
30 IFA>BTHENPRINT"NON !!!":GOTO1021
40 UNTILM$=LEFT$(M$(A),3)
50 F$=RIGHT$(M$(A),2):PRINTF$;
60 K=VAL("#"+F$):POKED,K:D=D+1
70 RETURN
100 F$=MID$(HEX$(PEEK(D)),2):D=D+1
110 IFLEN(F$)<2THENF$="0"+F$
120 IFF$="0"THENF$="00"
130 PRINT" "F$;
140 RETURN
200 GOSUB10:K=VAL(T$)
210 POKED,K:GOSUB100:PRINT"      ";
220 GOTO1020
300 GOSUB10:K=VAL(T$):DOKED,K
310 GOSUB100
320 GOSUB100:PRINT"   ";:GOTO1020
1000 CLS:PRINT:PRINTCHR$(4),,
1005 PRINTCHR$(27);
1006 PRINT"J  MINI-ASSEMBLEUR 6502";
1007 PRINTCHR$(4):PRINT:PRINT
1010 PRINTTAB(26):INPUT"ORG";D:R$=""
1020 PRINTR$
1021 PRINTHEX$(D);:INPUTT$
1025 L=LEN(T$):M$=LEFT$(T$,3)
1026 F$=RIGHT$(T$,1):R$=T$
1028 IFT$="ORG"THEN1010
1029 IFT$="NUL"THEND=D-1:GOTO1021
1030 IFT$="END"THENPRINT:END
1031 K=PEEK(616)-1:POKE616,K
1032 DOKE18,48000+40*K
1033 PRINTHEX$(D)"-    ";
1040 IFL=3THENA=-1:B=24:GOSUB10:PRINT"
      ";:GOTO1020
1050 IFL=5THENA=24:B=28:GOSUB10:PRINT"
      ";:GOTO1020
1060 IFL=7THENA=51:B=72:T$=MID$(T$,5,3):
GOTO200
1070 IFL=8THENA=103:B=114:T$=MID$(T$,6,3
):GOTO200
1080 IFL=11ANDM$="JMP"THENA=149:B=150:T$
=MID$(T$,6,5):GOTO300
1090 IFRIGHT$(T$,3)=")Y"THENA=131:B=139
:T$=MID$(T$,6,3):GOTO200
1100 IFF$=")"THENA=123:B=131:T$=MID$(T$,
6,3):GOTO200
1110 IFL=11ANDF$="X"THENA=88:B=103:TNEXT
=MID$(T$,5,5):GOTO300
1120 IFL=11ANDF$="Y"THENA=114:B=123:T$=M
ID$(T$,5,5):GOTO300
1130 IFF$="X"THENA=72:B=88:T$=MID$(T$,5,
3):GOTO200
1140 IFF$="Y"THENA=147:B=149:T$=MID$(T$,
5,3):GOTO200
1150 IFLEFT$(T$,1)="B"THEN1170
1160 A=28:B=51:T$=MID$(T$,5,5):GOTO300
1170 A=139:B=147:GOSUB10
1180 K=VAL(MID$(T$,5,5))
1190 IFK<DTHENK=255+K-D:GOTO210
1200 K=K-D-1:GOTO210
5000 CLS:PLOT14,12,"UN INSTANT..."
5010 FORR=0TO150:IFT$=""THENREADT$
5020 M$(R)=LEFT$(T$,5):T$=MID$(T$,6)
5025 NEXT:GOTO1000
5030 DATABRK00CLC18CLDD8CLI58CLVB8DEXCAD
EY88INXE8INYC8NOPEAPHA48PHP
08PLA68
5040 DATAPLP28RTI40RTS60SEC38SEDF8SEI78T
AXAATAYA8TSXBATXA8ATXS9ATYA
98ASL0A
5050 DATALSR4AROL2AROR6AADC6DAND2DASL0EB
IT2CCMPCDCPXECCPYCCDECCEEOR
4DINCEE
5060 DATAJMP4CJSR20LDAADLDXAELDYACLSR4EO
RA0DROL2EROR6ESBCEDSTA8DSTX
8ESTY8C
5070 DATAADC65AND25ASL06BIT24CMPC5CPXE4C
PYC4DECC6EOR45INCE6LDAA5LDX
A6LDYA4
5080 DATALSR46ORA05ROL26ROR66SBCE5STA85S
TX86STY84ADC74AND35ASL16CMP
D5DECD6
5090 DATAEOR55INCF6LDAB5LDYB4LSR56ORA15R
OL36ROR76SBCF5STA95STY94ADC
7DAND3D
5100 DATAASL1ECMPDDDECDEEOR5DINCFELDABDL
DYBCLSR5EORA1DROL3EROR7ESBC
FDSTA9D
5110 DATAADC69AND29CMPC9CPXE0CPYC0EOR49L
DAA9LDXA2LDYA0ORA09SBCE9ADC
79AND39
5120 DATACMPD9EOR59LDAB9LDXBEORA19SBCF9S
TA99ADC61AND21CMPC1EOR41LDA
A1ORA01
5130 DATASBCE1STA81ADC71AND31CMPD1EOR51L
DAB1ORA11SBCF1STA91BCC90BCS
B0BEQF0
5140 DATABMI30BNED0BPL10BVC50BVS70LDX86S
TX96JMP6C
```