9/8.19

Affichage permanent du contenu d'une mémoire

Voici un programme utilitaire bien pratique qui permet de surveiller l'évolution d'une case mémoire en temps réel, c'est-à-dire pendant l'exécution d'un autre programme.

Ce programme fait appel aux interruptions et au concept de RSX. Il est donc écrit en Assembleur.

COMMENT UTILISER LE PROGRAMME

Si vous désirez utiliser le programme sous sa forme Assembleur, saisissez le listing des pages suivantes :

```
 1                          ORG  9000H
 2                          LOAD 9000H
 3                ;
 4                ;---------------------------------
 5                ; RSX contenu d'une memoire
 6                ;---------------------------------
 7                ;
 8 9000 C32290              JP   DEFRSX          ;Definition RSX
 9                ;
10                ;---------------------------------
11                ; Declaration des constantes et
12                ; variables du programme
13                ;---------------------------------
14                ;
15 9003 00000000 EVBL:       DB   0,0,0,0
16 9007 00000000             DB   0,0,0,0
17 900B 00000000             DB   0,0,0,0,0
17 900F 00
18                SAVA:      DS   1               ;Sauvegarde de A
19                SAVCUR:    DS   2               ;Sauv coord curseur
20                BUF:       DS   4               ;Zone RAM pour LOG EXT
21 9017 1C90      PTRTAB:    DW   TABLE           ;Pointeur TABLE
22 9019 C32F90             JP   TRAITE          ;Traitement
23 901C 5641      TABLE:     DB   "VA"
24 901E D2                 DB   "R"+80H
25 901F 00                 DB   0               ;Fin de table
26 9020 0100      VAR:       DW   1               ;Adresse de la variable
27                ;
28                INITEV:    EQU  0BCEFH          ;INIT EVEN BLOC
29                ADDEVE:    EQU  0BCE9H          ;ADD EVEN BLOC
30                TXTOUT:    EQU  0BB5AH          ;TXT OUTPUT
```

```
31                      SETCUR:     EQU  0BB75H          ;TXT SET CURSOR
32                      GETCUR:     EQU  0BB78H          ;TXT GET CURSOR
33                      CUREN:      EQU  0BB7BH          ;TXT CUR ENABLE
34                      CURDIS:     EQU  0BB7EH          ;TXT CUR DISABL
35                      LOGEXT:     EQU  0BCD1H          ;KL LOG EXT
36                      ;
37                      ;--------------------------------
38                      ; Definition de la RSX
39                      ;--------------------------------
40                      ;
41                      DEFRSX:     EQU  $               ;Point d'entree
42 9022 011790                      LD   BC,PTRTAB       ;Ptr table definition
43 9025 211390                      LD   HL,BUF          ;Buffer pour LOG EXT
44 9028 CDD1BC                      CALL LOGEXT          ;Definition de la RSX
45 902B CD3C90                      CALL INSTALL         ;Installation de l'IT
46 902E C9                          RET
47                      ;
48                      ;--------------------------------
49                      ; Traitement de la RSX
50                      ; Validation ou devalidation
51                      ; de l'affichage, saisie de l'@
52                      ;--------------------------------
53                      ;
54                      TRAITE:     EQU  $
55 902F DD7E00                      LD   A,(IX+0)
56 9032 322090                      LD   (VAR),A
57 9035 DD7E01                      LD   A,(IX+1)
58 9038 322190                      LD   (VAR+1),A
59 903B C9                          RET
```

```
60                      ;
61                      ;
62                      ;--------------------------------
63                      ; Installation de l'IT
64                      ;--------------------------------
65                      ;
66                      INSTALL:    EQU  $
67 903C 210990                      LD   HL,EVBL+6
68 903F 0681                        LD   B,81H
69 9041 0E00                        LD   C,0
70 9043 115690                      LD   DE,TRAIT          ;@ Traitement
71 9046 CDEFBC                      CALL INITEV            ;INIT EVEN BLOC
72 9049 210390                      LD   HL,EVBL
73 904C 110100                      LD   DE,1
74 904F 011900                      LD   BC,25
75 9052 CDE9BC                      CALL ADDEVE            ;ADD EVEN BLOC
76 9055 C9                          RET
77                      ;
78                      ;--------------------------------
79                      ; Routine de traitement
80                      ;--------------------------------
81                      ;
82                      TRAIT:      EQU  $
83 9056 F3                          DI
84 9057 F5                          PUSH AF
85 9058 C5                          PUSH BC
86 9059 D5                          PUSH DE
87 905A E5                          PUSH HL
88 905B DDE5                        PUSH IX
89 905D FDE5                        PUSH IY
90                      ;
```

```
 91 905F 2A2090             LD   HL,(VAR)
 92 9062 7C                 LD   A,H
 93 9063 B5                 OR   L
 94 9064 CAB290             JP   Z,FINIT          ;Pas d'affichage
 95                ;
 96 9067 CD78BB             CALL GETCUR           ;Position courante curs
 97 906A 221190             LD   (SAVCUR),HL
 98 906D CD7EBB             CALL CURDIS           ;TXT CUR DISABLE
 99 9070 2601               LD   H,1              ;Colonne curseur
100 9072 2E01               LD   L,1              ;Ligne curseur
101 9074 CD75BB             CALL SETCUR           ;Position curseur
102               ;
103 9077 2A2090             LD   HL,(VAR)
104 907A 7E                 LD   A,(HL)
105 907B CB3F               SRL  A
106 907D CB3F               SRL  A
107 907F CB3F               SRL  A
108 9081 CB3F               SRL  A
109 9083 FE0A               CP   10
110 9085 3004               JR   NC,MSBLET
111 9087 C630               ADD  A,48
112 9089 1802               JR   FAIBLE
113               MSBLET:   EQU  $
114 908B C637               ADD  A,55
115               FAIBLE:   EQU  $
116 908D CD5ABB             CALL TXTOUT           ;Aff MSB
117 9090 7E                 LD   A,(HL)
118 9091 E60F               AND  0FH
119 9093 FE0A               CP   10
```

```
120 9095 3004              JR   NC,LSBLET
121 9097 C630              ADD  A,48
122 9099 1802              JR   FINHEX
123              LSBLET:   EQU  $
124 909B C637              ADD  A,55
125              FINHEX:   EQU  $
126 909D CD5ABB            CALL TXTOUT          ;Aff LSB
127              ;
128 90A0 3E20              LD   A,32
129 90A2 CD5ABB            CALL TXTOUT          ;Espacement
130 90A5 7E                LD   A,(HL)
131 90A6 CD5ABB            CALL TXTOUT          ;Affichage caractere
132              ;
133              FINAFF:   EQU  $
134 90A9 2A1190            LD   HL,(SAVCUR)
135 90AC CD75BB            CALL SETCUR          ;Restitution curseur
136 90AF CD7BBB            CALL CUREN           ;TXT CUR ENABLE
137              ;
138              FINIT:    EQU  $
139 90B2 FDE1              POP  IY
140 90B4 DDE1              POP  IX
141 90B6 E1                POP  HL
142 90B7 D1                POP  DE
143 90B8 C1                POP  BC
144 90B9 F1                POP  AF
145 90BA FB                EI
146 90BB C9                RET
147                        END
```

```
ADDEVE      BCE9 BUF        9013 CUREN      BB7B CURDIS     BB7E
DEFRSX      9022 EVBL       9003 FAIBLE     908D FINHEX     909D
FINAFF      90A9 FINIT      90B2 GETCUR     BB78 INITEV     BCEF
INSTALL     903C LOGEXT     BCD1 LSBLET     909B MSBLET     908B
PTRTAB      9017 SAVA       9010 SAVCUR     9011 SETCUR     BB75
TABLE       901C TXTOUT     BB5A TRAITE     902F TRAIT      9056
VAR         9020
```

Assemblez-le et initialisez la RSX en tapant sous Basic :

CALL &9000

Si vous préférez utiliser un chargeur Basic, en voici le listing et les données de checksum correspondantes :

```
10 REM ------------------------------------------
20 REM Affichage permanent du contenu d'une memoire
30 REM ------------------------------------------
40 REM
50 FOR i=&9000 TO &90BB
60   READ a$
70   a=VAL("&"+a$)
80   POKE i,a
90 NEXT i
100 END
110 REM ------------------------------------------
1000 DATA C3,22,90,0,0,0,0,0,0,0,0,0,0,0,0,0
1010 DATA 0,0,0,0,0,0,0,1C,90,C3,2F,90,56,41,D2,0
1020 DATA 1,0,1,17,90,21,13,90,CD,D1,BC,CD,3C,90,C9,DD
1030 DATA 7E,0,32,20,90,DD,7E,1,32,21,90,C9,21,9,90,6
1040 DATA 81,E,0,11,56,90,CD,EF,BC,21,3,90,11,1,0,1
1050 DATA 19,0,CD,E9,BC,C9,F3,F5,C5,D5,E5,DD,E5,FD,E5,2A
1060 DATA 20,90,7C,B5,CA,B2,90,CD,78,BB,22,11,90,CD,7E,BB
1070 DATA 26,1,2E,1,CD,75,BB,2A,20,90,7E,CB,3F,CB,3F,CB
1080 DATA 3F,CB,3F,FE,A,30,4,C6,30,18,2,C6,37,CD,5A,BB
1090 DATA 7E,E6,F,FE,A,30,4,C6,30,18,2,C6,37,CD,5A,BB
1100 DATA 3E,20,CD,5A,BB,7E,CD,5A,BB,2A,11,90,CD,75,BB,CD
1110 DATA 7B,BB,FD,E1,DD,E1,E1,D1,C1,F1,FB,C9,0,0,0,0
```

```
76 9A D 2D
C9 94 BE 90 7A A4 3D 4
```

Initialisez la RSX en tapant :

```
CALL &9000
```

La RSX est maintenant prête à être utilisée.

Si vous désirez suivre l'évolution de la case mémoire située en &8000, tapez :

```
|VAR,&8000
```

Les valeurs hexadécimales et ASCII de cette case mémoire sont affichées en haut et à gauche de l'écran. Modifiez le contenu de cette case mémoire, en tapant par exemple :

```
POKE &8000,65
```

L'affichage en haut de l'écran est immédiatement révisé et correspond à la valeur pokée.

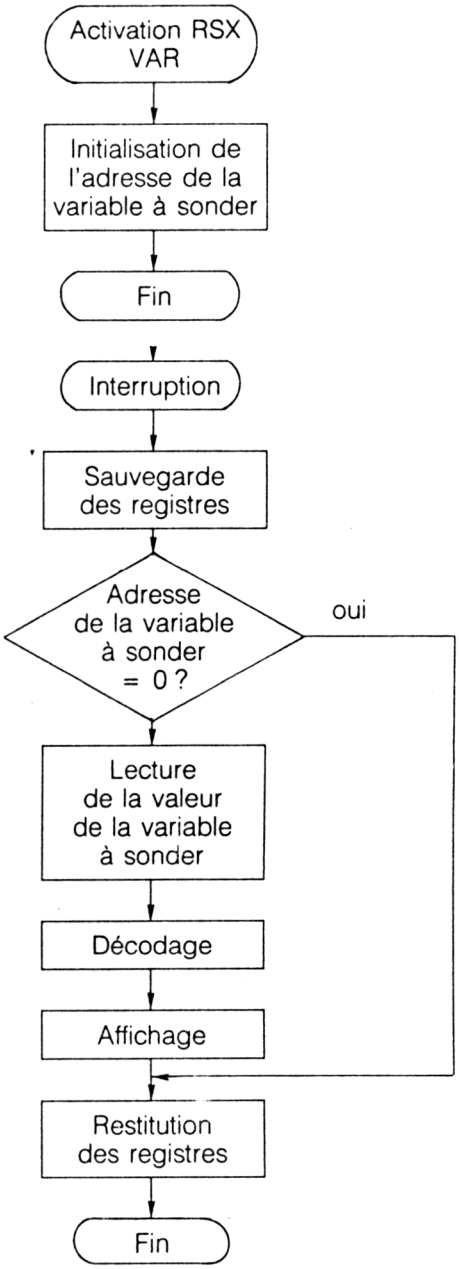
Si vous désirez dévalider le suivi de cette case mémoire, tapez simplement :

```
|VAR,0
```

Cette RSX peut évidemment être utilisée en parallèle d'un autre programme, et c'est là tout son intérêt. Elle pourra servir par exemple pour la mise au point de programmes Assembleur ou Basic.

LE PROGRAMME EN DÉTAIL

La logique du programme obéit à l'ordinogramme suivant :



Avant de pouvoir utiliser la RSX ¦ VAR, il faut l'installer. Le court programme qui commence à l'étiquette **DEFRSX** réalise cette installation. Pour ce faire, il utilise la macro **LOGEXT** :

```
DEFRSX:   EQU    $
          LD     BC,PTRTAB
          LD     HL,BUF
          CALL   LOGEXT
```

Avant d'appeler cette macro, il convient de placer dans :

— le registre **BC** l'adresse de la table de commande de la RSX,

— le registre **HL** l'adresse d'un buffer de 4 octets utilisé de manière interne par le FIRMWARE.

Comme cette RSX utilise une interruption, la phase d'initialisation doit également comporter la définition de l'interruption :

```
CALL INSTALL
```

Le sous-programme INSTALL active séquentiellement deux macros du FIRMWARE. La macro **INITEV** initialise le bloc d'interruption :

```
INSTALL:  EQU    $
          LD     HL,EVBL+6
          LD     B,81H
          LD     C,0
          LD     DE,TRAIT
          CALL   INITEV
```

Avant d'appeler la macro **INITEV**, les registres suivants doivent être initialisés :

— **HL** = Adresse du bloc d'interruption,

— **B** = Type d'interruption (asynchrone dans notre cas),

— **C** = Adresse de ROM select de la routine d'interruption (0 dans notre cas),

— **DE** = Adresse de la routine d'interruption (TRAIT dans notre cas).

Lorsque le bloc d'interruption est initialisé, il faut encore définir la fréquence des interruptions. C'est le but de la macro **ADDEVE** :

```
LD      HL,EVBL
LD      DE,1
LD      BC,25
CALL    ADDEVE
```

Avant d'appeler cette macro, les registres suivants doivent être initialisés :

— **HL** = Adresse du bloc d'interruption,

— **DE** = Valeur initiale du compteur,

— **BC** = Valeur finale du compteur.

La différence entre **BC** et **DE** représente un intervalle de temps en 1/50 secondes (25/50 sec. soit 0.5 sec. dans notre cas). La routine d'interruption sera activée chaque fois qu'un tel intervalle de temps se sera écoulé, c'est-à-dire deux fois par seconde dans notre cas.

Cette phase d'initialisation exécutée, la RSX peut maintenant être utilisée :

Lorsqu'une instruction du type :

|VAR,<Adresse>

est rencontrée par l'interpréteur, il examine la liste des RSX en mémoire, et exécute la routine de traitement correspondante. Dans notre cas, la routine située à l'étiquette **TRAITE**.

Cette routine se contente de récupérer le paramètre qui lui est passé et de le stocker dans la variable **VAR**.

```
TRAITE:   EQU     $
          LD      A,(IX+0)
          LD      (VAR),A
          LD      A,(IX+1)
          LD      (VAR+1),A
          RET
```

Deux fois par seconde, la routine située à l'étiquette **TRAIT** est exécutée. Les premières actions effectuées par cette routine consistent à dévalider les interruptions et à sauvegarder les registres dans la pile :

```
TRAIT :   EQU     $
          DI
          PUSH    AF
          PUSH    BC
          PUSH    DE
          PUSH    HL
          PUSH    IX
          PUSH    IY
```

L'adresse mémoire à sonder, c'est-à-dire la valeur qui se trouve dans la variable **VAR** est ensuite testée. Par convention, si cette valeur est nulle, aucun traitement n'est effectué.

```
LD      HL,(VAR)
LD      A,H
OR      L
JP      Z,FINIT
```

Dans le cas contraire, le contenu de la mémoire à sonder doit être affiché en haut et à gauche de l'écran. Pour ne pas perturber l'affichage, le curseur est éteint. Sa position courante est sauvegardée pour pouvoir la restituer après l'affichage :

```
CALL    GETCUR
LD      (SAVCUR),HL
CALL    CURDIS
```

Le curseur est ensuite positionné en haut et à gauche de l'écran :

```
LD      H,1
LD      L,1
CALL    SETCUR
```

Le contenu de la mémoire à examiner est ensuite affiché à l'aide de la macro **TXTOUT**. Cette macro affiche le caractère dont le code ASCII lui est passé. Il est donc nécessaire de décomposer et de convertir le poids fort et le poids faible de la valeur à afficher.

L'extraction du poids fort se fait à l'aide d'un décalage logique vers la droite :

```
LD      HL,(VAR)
LD      A,(HL)
SRL     A
SRL     A
SRL     A
SRL     A
```

Si la valeur obtenue est supérieure ou égale à 10, il s'agit d'une lettre. La conversion ASCII est obtenue en ajoutant 55 :

```
CP      10
ADD     A,55
```

Dans le cas contraire, il s'agit d'un chiffre. La conversion ASCII est obtenue en ajoutant 48 :

```
ADD     A,48
```

Dans tous les cas, le poids fort est affiché à l'aide de la macro **TXTOUT** :

```
CALL    TXOUT
```

L'extraction du poids faible est plus simple. Il suffit d'effectuer un **ET** logique sur l'octet concerné :

```
LD      A,(HL)
AND     0FH
```

La conversion du poids faible en le code ASCII correspondant passe par les mêmes étapes que précédemment.

Lorsque le contenu de la mémoire à sonder est affiché sur l'écran, son code ASCII est également affiché :

```
LD      A,32
CALL    TXTOUT
LD      A,(HL)
CALL    TXTOUT
```

La routine de traitement se termine par la restitution du curseur (position et affichage) :

```
LD      HL,(SAVCUR)
CALL    SETCUR
CALL    CUREN
```

par la restitution des registres sauvegardés et la revalidation des interruptions :

```
POP     IY
POP     IX
POP     HL
POP     DE
POP     BC
POP     AF
EI
RET
```