Logiciel de permutation des pages Ø et 1 du 6502

JSR SWAP

Lorsque deux (ou plusieurs) programmes d'envergure doivent coexister en mémoire du 6502, comme par exemple un interpréteur BASIC et un DOS, ou encore les deux précédents avec un logiciel vidéo, le conflit est inévitable en page zéro et sur la pile (en page 1). Un procédé de conciliation classique consiste à dédoubler ces deux pages en mémoire vive, par exemple en E000...E0FF pour la page zéro et en E100...E1FF pour la pile. On permute alors le contenu de ces zones de mémoire vive et celui des pages 0 et 1 chaque fois que l'on passe d'un programme à l'autre. De sorte qu'il ne subsiste aucun risque de destruction mutuelle de pointeurs en page zéro ou du contenu de la pile.

Le microprocesseur 6502 est caractérisé, entre autres, par l'usage qu'il fait des pages 0 et 1. Les 256 octets de 0000_{HEX} à 00FF peuvent être adressés à l'aide de codes opératoires spécifiques à cette zone (adressage en page zéro: l'octet d'adresse de poids fort n'est pas spécifié lors d'une manipulation de donnée dans cette zone; il est implicite dans le code opératoire). Ces mêmes 256 octets peuvent être utilisés comme pointeurs de 16 bits, pour un adressage indirect indexé du reste de la mémoire. Les 256 octets de 0100_{HEX} à 01FF constituent la pile du 6502, un registre de sauvegarde géré par le processeur lui-même, caractérisé par sa structure "dernier entré, premier sorti": le processeur n'est, en principe, capable de manipuler que le dernier article entré dans la pile. Il dispose pour cela d'un pointeur de pile interne.

On comprend aisément que la moindre altération intempestive d'un paramètre sauvegardé dans l'une de ces deux pages compromettra — le plus souvent irrémédiablement — le bon déroulement d'un programme en cours d'exécution. Lorsque deux programmes sont exécutés parallèlement, il est donc essentiel qu'ils ne se détruisent pas mutuellement leurs paramètres en page 0 et 1. Pour le programmeur, c'est un souci supplémentaire, et parfois un problème insoluble. Dès que les programmes en présence ont une certaine envergure, il est préférable de s'affranchir radicalement de ces servitudes qui entravent l'usage des pages 0 et 1: en termes de diplomatie, le procédé utilisé s'appellerait "l'extra-territorialité"...

Avec la routine proposée ici, on transfère le contenu de la page zéro et de la page un vers une autre zone de mémoire vive pour les y mettre à l'abri de toute altération intempestive. Simultanément, le contenu de la zone de mémoire vive en question est transféré dans les pages zéro et un. En jargon de programmateur, cela s'appelle une permutation (*swap* en anglais).

On n'a donc plus à se soucier du contenu de ces deux pages au moment où l'on quitte un programme pour en exécuter un autre. Il suffit d'exécuter la routine SWAP, et le tour est joué: la page zéro (0000_{HEX}...$00FF_{HEX}$) et la page un (0100_{HEX}...$01FF_{HEX}$) du premier programme sont sauvegardées en $E000_{HEX}$...$E1FF_{HEX}$, tandis que le contenu des pages zéro et un spécifique au second programme, sauvegardé jusque-là en $E000_{HEX}$...$E1FF_{HEX}$, est transféré en 0000_{HEX}...$01FF_{HEX}$. Lorsque l'on repasse du second au premier programme, on exécute à nouveau la routine SWAP, et c'est l'opération inverse qui a lieu. Bien entendu, les addresses $E000_{HEX}$...$E1FF_{HEX}$ peuvent être modifiées en fonction des exigences du système sur lequel la routine SWAP sera utilisée. En tous cas, **il s'agit toujours d'une zone de mémoire vive**. La routine SWAP elle-même doit également toujours se trouver en mémoire vive. Il suffit de jeter un rapide coup d'oeil à la dernière ligne du listing pour comprendre pourquoi le sous-programme SWAP ne peut fonctionner qu'en mémoire vive! Indexation et permutation sont les deux mammelles de la programmation, ne l'oubliez pas!

```
0040:                   ******
0050:                   *SWAP*
0060:                   ******
0070:
0080:
0090:                   SWAP PAGE 0 AND PAGE 1 WITH E000 AND E100
0100:
0110:
0120:
0130: E200              PZ      *       $0000   PAGE ZERO
0140: E200              STACK   *       $0100   STACK AREA
0150: E200              SPZ     *       $E000   SWAPPED PAGE ZERO
0160: E200              SSTACK  *       $E100   SWAPPED STACK AREA
0170:
0180:
0190:
0200: E200 18           SWAP    CLC
0210: E201 68                   PLA
0220: E202 69 01                ADCIM $01       PUT RETURN ADDRESS
0230: E204 8D 2C E2             STA   JMPINS +01 JUST BEHIND A JUMP OP-CODE
0240: E207 68                   PLA
0250: E208 AA                   TAX
0260: E209 90 01                BCC   SW
0270: E20B E8                   INX
0280:
0290: E20C 8E 2D E2     SW      STX   JMPINS +02
0300: E20F A2 00                LDXIM $00       RESET INDEX
0310:
0320: E211 BD 00 01     SWAPST  LDAX  STACK   GET BYTE FROM PAGE 1
0330: E214 BC 00 E1             LDYX  SSTACK  GET BYTE FROM SWAP AREA
0340: E217 9D 00 E1             STAX  SSTACK  SAVE BYTE FROM PAGE 1 IN SWAP AREA
0350: E21A 98                   TYA
0360: E21B 9D 00 01             STAX  STACK   SAVE BYTE FROM SWAP AREA IN PAGE 1
0370: E21E B5 00                LDAX  PZ      GET BYTE FROM PAGE 0
0380: E220 BC 00 E0             LDYX  SPZ     GET BYTE FROM SWAP AREA
0390: E223 9D 00 E0             STAX  SPZ     SAVE BYTE FROM PAGE 0 IN SWAP AREA
0400: E226 94 00                STYZX PZ      SAVE BYTE FROM SWAP AREA IN PAGE 0
0410: E228 E8                   INX           BUMP THE OFFSET
0420: E229 D0 E6                BNE   SWAPST NOT DONE, KEEP ON
0430:
0440: E22B 4C FF FF     JMPINS  JMP   $FFFF   SELF MODIFYING CODE!!!
0450:
```

Tableau 1. On quitte la routine de permutation (à laquelle il a été fait appel à l'aide d'une instruction JSR) non par un RTS, mais par un JMP! L'adresse de retour est "dépilée" au début de la routine, puis corrigée (adresse de retour = adresse de départ + 1) et enfin placée derrière l'instruction de saut au label JMPINS.