swap-routine voor 6502
elektuur juni 1985

een verwisselroutine voor page 0 en 1

Als twee (of meer) programma's van hetzelfde nivo in het geheugen van de 6502 zijn opgeslagen, is het onvermijdbaar dat op een zeker moment iets misgaat in page zero en de stack op page 1. Denk maar aan de kombinatie van een BASIC-interpreter en een DOS, of die twee met daarbij ook nog eens een video-gedeelte. Een klassieke oplossing hiervoor is het kopiëren van die twee "pagina's" in het RAM-geheugen, bijvoorbeeld op E000...E0FF voor page 0 en op E100...E1FF voor de stack. Telkens als men dan overspringt van het ene programma naar het andere, worden die gedeeltes in het RAM-bereik verwisseld met page 0 en 1. Op die manier is men ervan verzekerd dat voor geen van de programma's de pointers in page 0 of de inhoud van de stack verloren kunnen gaan.

swap-routine voor 6502

```
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:
```

Tabel 1. De swap-routine. De routine wordt niet verlaten via een RTS, maar via een JMP! Het terugkeeradres wordt opgepikt aan het begin van de routine, vervolgens gekorrigeerd (terugkeeradres = vertrek-adres + 1) en dan achter de sprong-instruktie bij label JMPINS gezet.

Een van de kenmerken van de 6502-microprocessor is de indeling van de pages 0 en 1 (een fatsoenlijk Nederlands woord kennen we hiervoor eigenlijk niet). De 256 bytes van 0000 tot 00FF kunnen worden geadresseerd met behulp van specifieke opcodes voor dat bereik (adressering op page 0: het meest significante adres-byte hoeft niet te worden aangegeven bij een data-bewerking in dit bereik; dat is al verwerkt in de opcode). Diezelfde 256 bytes kunnen ook worden gebruikt als 16-bits pointers voor een indirekt geïndexeerde adressering in het overige geheugenbereik.
De 256 bytes van 0100 tot 01FF bevatten de stack van de 6502, een opslagregister dat door de processor zelf beheerd wordt. Dat laatste werkt volgens het systeem "last in, first out": de processor is in principe alleen maar in staat om het laatste gegeven dat op de stack is gezet te manipuleren. Daartoe beschikt hij over een interne stack pointer.
Het is eenvoudig in te zien dat ook maar de kleinste verandering van een parameter in een van deze twee pages ernstige gevolgen kan hebben voor het verloop van het programma dat op dat moment wordt uitgevoerd. Als twee programma's tegelijkertijd "draaien", is het bijzonder belangrijk dat ze niet elkaars data op pagina 0 en 1 beïnvloeden of zelfs geheel "verwoesten". Voor de programmeur is dat een extra punt waar hij op moet letten, vaak vormt dat zelfs een onoplosbaar probleem. Als de aanwezige programma's wat uitgebreider en/of ingewikkelder zijn, wordt het praktisch ondoenlijk om alles nog in het oog te houden en in goede banen te leiden. Dan wordt het tijd om gebruik te maken van een subroutine die gewoon het hele handeltje van page 0 en 1 in een ander geheugenbereik opslaat. Door deze subroutine worden de data van page 0 en 1 dus weggeschreven in een bereik waar ze veilig zijn, zolang er wordt gewerkt met een ander programma. Bovendien zet de subroutine de oude inhoud van dat RAM-gebied weer in page 0 en 1. In het Engelse vakjargon heet dat een **swap**.
Men hoeft zich zo geen zorgen meer te maken over de inhoud van page 0 en 1 als wordt overgeschakeld van het ene naar het andere programma. Wel moet tijdens dat overschakelen steeds even naar de swap-routine worden gesprongen, die dan het omwisselen van page 0 en 1 verzorgt. Pagina 0 en 1 (0000...01FF) worden dus gekopieerd op E000...E1FF en de gegevens voor het nieuwe programma, die waren opgeborgen op E000...E1FF, worden weer teruggezet op 0000...01FF. Als men overspringt van het tweede naar het eerste programma, wordt ook weer de swap-routine uitgevoerd, die de data in de twee bewuste geheugenblokken opnieuw verwisselt. We willen nog even opmerken dat men ook een ander geheugenbereik dan E000...E1FF voor het kopiëren kan kiezen, afhankelijk van het gebruikte computersysteem waarop de swap-routine gebruikt wordt. Het is alleen belangrijk dat dit bereik in het RAM-gedeelte ligt (anders kan er niet in worden geschreven door de processor). De swap-routine zelf moet ook altijd in RAM staan. Dat wordt wel duidelijk als u naar de laatste regel van de listing kijkt!