special

D. M. DE BOER

Zero page shifter

U heeft het vast wel eens bij de hand gehad, u wilt in programma A een programma B aanroepen. Het probleem kan zich voordoen dat beide programma's (gedeeltelijk) dezelfde zero page locaties gebruiken. Of u gebruikt zero page locaties die ook door de monitor worden gebruikt, waardoor debuggen onmogelijk wordt. De enige oplossing is dan, om alle zero page adressen naar wens aan te passen. Wanneer u met een assembler werkt is dat een koud kunstje. Wanneer het omzetten met de hand moet gebeuren is het een vervelend en tijdrovend werk met grote kans op fouten. De 'zero page shifter' doet het voor u! Hij selecteert in een fractie van een seconde de opcodes die betrekking hebben op een zero page adres, en vervangt dan (indien gewenst) het bijbehorende adres.

Gebruik

Natuurlijk moet u voordat u het programma start wat gegevens over het te modificeren programma invoeren. Allereerst het beginadres. De eerste twee cijfers (high order) van dit adres komen op $ 00E2, de laatste twee cijfers (low order) op adres $ 00E1. Het einde van het te modificeren programma dient u te markeren met $ FF.
Voor het veranderen van de zero page adressen maakt het programma gebruik van twee tabellen. In de eerste tabel geeft u aan welke adressen moeten worden veranderd. Deze tabel loopt van adres 00D1...00E0. U kunt in één keer 16 adressen veranderen. Indien er minder adressen moeten worden vervangen dient u de rest van de tabel te vullen met '00'. In de tweede tabel (00C1... 00D0) noteert u, op de overeenkomstige plaats, welk adres in plaats van het oude adres moet komen.

Voorbeeld

Als voorbeeld nemen we de subroutine 'LETTER' welke we in RB no. 5, 6 en 7 1979 uitvoerig hebben besproken. Stel dat deze subroutine wordt aangeroepen door een hoofdprogramma die alle zero page locaties boven $ C4 nodig heeft. We moeten de zero page locaties DO. . . D5 (welke door 'LETTER' worden gebruikt) naar een gebied brengen waar het hoofdprogramma geen 'last' heeft van deze subroutine. In ons voorbeeld nemen we hiervoor A8. . . AD, welke nog vrij zijn. Allereerst voeren we het beginadres ($ 0600) in door '00' te zetten op adres $ 00E1, en '06' op adres $ 00E2.
Vervolgens markeren we het eind van het programma door 'FF' te zetten op adres $ 0883 (dit is de eerste vrije locatie ná het programma). Tot slot vullen we de tabellen in: Vanaf adres 00D1 de te veranderen adressen: D0, D1, D2, D3, D4, D5 en vervolgens 10 maal '00' om de tabel vol te maken. Vanaf adres 00C1 komen nu de 'nieuwe' adressen, A3, A9, AA, AB, AC, AD en weer 10 maal '00', om ook deze tabel vol te maken. Alle gegevens zijn nu ingevoerd, en de zero page shifter kan worden gestart op 1780. Bij korte programma's zal het net lijken of er niets gebeurt, bij iets langere programma's dooft het display voor korte tijd. In ieder geval zult u zien dat de zero page shifter zijn (haar?) taak feilloos volbracht.

Waarschuwing

Na het starten van de zero page shifter zal de pointer (00E1 en 00E2) aan het einde van het te modificeren programma staan. Druk daarom niet voor de tweede maal op 'GO', want de zero page shifter gaat dan de geheugenruimte ná het programma modificeren hetgeen tot ongewenste resultaten kan leiden. Wanneer een programma wordt gevolgd door een tabel moet de stopcode geplaatst worden tussen tabel en programma. Ook kan het voorkomen dat een tabel in pagina 0 moet worden verplaatst. Vaak wordt een tabel geadresseerd met een adres dat één lager ligt dan het beginadres van de tabel bijv. LDA TAB-1, X. In dit geval moet de waarde TAB-1 als te veranderen adres worden opgegeven.

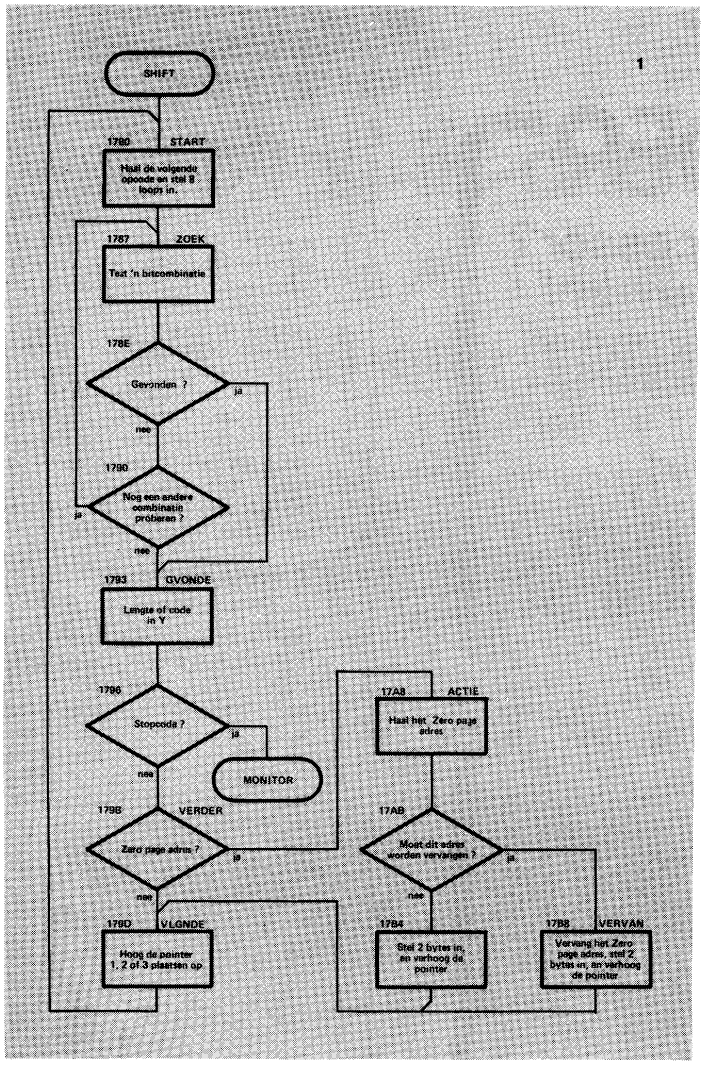
Werking

Om dit programma naar behoren uit te voeren moet het programma weten welke geheugenplaatsen een opcode bevatten en welke adressen een operand (adres) bevatten. Hiervoor is het noodzakelijk dat voor elke opcode de lengte van de instructie wordt bepaald. Het principe van deze lengte bepaling is min of meer afgekeken van het programma 'relocate' (first book of KIM). Nadat de eerste opcode in de registers Y en A is gezet begint de lengte bepaling. De opcode blijft gedurende dit proces 'onbeschadigd' in register Y staan. De kopie van de opcode in register A wordt steeds bewerkt met gege-

Tabel1

tabel 1 Een overzicht van de geteste bitcombinaties met de bijbehorende instructies.

Register X	Opcode	Instructies	Lengte/code
8	XXXX XX11	stopcode	FF
7	XXXX 11XX	abs en abs,X	03
6	XXX1 1001	abs,Y	03
5	0010 0000	JSR	03
4	XXXX 10X0	impl en accu	01
3	01X0 0000	RTI en RTS	01
2	XXXX 01XX	Zp Zp,X en Zp,Y	00
1	XXXX 0001	(ind,X) en (ind),Y	00
0	overblijvende	relatief, imidiate	02



vens uit de tabellen MASK en BITS. Hierbij houdt index register X bij welke 'test' er op de opcode is uitgevoerd. Als voorbeeld nemen we X = 6. Op adres 1788 wordt hierdoor de opcode gemaskeerd met de waarde $ 1F (de 6e waarde uit MASK). We kijken dus alleen naar de laatste 5 bits van de opcode. Op adres 178B wordt de EOR bewerking uitgevoerd tussen de gemaskeerde opcode en '0001 1001' (de 6e waarde uit BITS). Dit wil zeggen dat de bits 0, 3 en 4 worden geïnverteerd. Alléén als deze bewerking '$ 00' oplevert zal de test worden beëindigd, anders gaan we door met X = 05. Welke opcodes hebben we nu geselecteerd? De eerste 3 bits werden al met het markeren op 0 gezet, dus hun aanvankelijke waarde doet er niet toe. Alle opcodes met de structuur 'XXX1 1001' leveren bij deze test '00' op, zodat we met X = 6 belanden op adres 1793. Hier wordt ook weer de 6e waarde (nu telt '0' ook mee) uit de tabel in register Y gezet. In dit geval is dat dus 03, de lengte van de geselecteerde opcodes (in dit geval alle ABS,Y instructies). Op deze wijze voert het programma maximaal 8 tests uit. Wanneer na de 8e test nog niets is gevonden weten we zeker dat de instructie 2 bytes lang is, omdat dan alle 1 en 3 bytes instructies alsmede een deel van de 2 bytes instructies zijn uitgefilterd. Wanneer een bitcombinatie is herkend wordt steeds op adres 1793 de lengte van de instructie in register Y gezet. Alleen bij de stopcode komt er $ FF, en bij een zero page instructie $ 00 in register Y. Hierdoor kunnen deze 2 gevallen onderscheiden worden van de andere opcodes. Zolang er niets hoeft te worden gemodificeerd, wordt op adres 179D de pointer afhankelijk van de waarde in Y, één, twee of drie plaatsen opgehoogd. Hierdoor zal hij weer precies de volgende opcode aanwijzen, en het hele proces begint opnieuw. Bij de stop-

Lijst 1

```
1780  A0 00     START   LDY   = $00
1782  B1 E1             LDA   (POINTL),Y   ┐ Haal de volg. opcode
1784  A8               TAY                 ┘ en stel 8 loops in
1785  A2 08             LDX   = $08
1787  98       ZOEK    TYA
1788  3D BF 17         AND   MASK-1,X     ┐ Test 'n bitcomb natie
178B  5D C7 17         EOR   BITS-1,X     ┘
178E  F0 03            BEQ   GVONDE       — Gevonden?
1790  CA               DEX                ┐ Nog een andere combi-
1791  D0 F4            BNE   ZOEK         ┘ nat e proberen?
1793  BC D0 17 GVONDE  LDY   CODE,X       — Lengte of code in Y
1796  10 03            BPL   VERDER       ┐ Indien stopcode
1798  4C 22 1C         JMP   RST          ┘ terug naar monitor
179B  F0 0B      VERDER  BEQ   ACTIE        — Indien Zpage act e!
179D  E6 E1      VLGNDE  INC   POINTL      ┐
179F  D0 02              BNE   $02          │ Hoog de pointer 1,2
17A1  E6 E2              INC   POINTH       │ of 3 plaatsen op
17A3  88                 DEY                ┘
17A4  D0 F7              BNE   VLGNDE       —
17A6  F0 D8              BEQ   START        — Onvoorwaardelijk terug
17A8  C8         ACTIE   INY                ┐ Haal het Zpage adres
17A9  B1 E1              LDA   (POINTL),Y   ┘
17AB  A2 10              LDX   = $10        ┐
17AD  D5 D0      ZOEK    CMP   ZPIN,X       │ Kijk of dit adres
17AF  F0 07              BEQ   VERVAN       │ moet worden vervangen?
17B1  CA                 DEX                │
17B2  D0 F9              BNE   ZOEK         ┘
17B4  A0 02              LDY   =$02         ┐ Stel 2 bytes in,en
17B6  D0 E5              BNE   VLGNDE       ┘ verhoog de pointer
17B8  B5 C0      VERVAN  LDA   ZPUIT,X      ┐ Vervang het Zpage
17BA  91 E1              STA   (POINTL),Y   ┘ adres
17BC  A0 02              LDY   =$02         ┐ Stel 2 bytes in,en
17BE  D0 DD              BNE   VLGNDE       ┘ verhoog de pointer
17C0  0F 0C 0F 0D FF 1F 0C 03      MASK
17C8  01 04 40 08 20 19 0C 03      BITS
17D0  02 00 00 01 01 03 03 03 FF   CODE
```

code wordt zoals al gezegd index register Y geladen met FF, de enige keer dat een negatief getal wordt geladen. Hierdoor springen we op adres 1796 niet over de 'JMP' heen, en keren we terug naar de monitor. Wanneer de opcode een zero page adres was wordt register Y met '00' geladen, zodat we op adres 179B met een BEQ naar 'ACTIE' kunnen springen. Het programma is verder niet meer ingewikkeld, eerst wordt het zero page adres in register A gekopieerd. Vervolgens wordt vanaf adres 17AB gekeken of dit adres voorkomt in de tabel met te veranderen adressen. Indien dit zo is, wordt het adres vervangen. Voordat we verder gaan met de volgende opcode moet eerst register Y met '02' worden geladen, om aan te geven dat de pointer met twee plaatsen moet worden verhoogd. Een sprong naar 'VLGNDE ($ 179D) zorgt dan dat we met de volgende opcode gaan beginnen.

Andere adressen, ander systeem

Wanneer de zero page shifter in een andere geheugenruimte moet werken, moeten de instructies op 1788, 178B en 1793 worden aangepast. Bij andere systemen moet (bovendien) de sprong naar de monitor worden gewijzigd (adres 1798). Uiteraard kan dit programma alleen door 65XX processoren worden uitgevoerd.