ZERO PAGE SHIFTER

D.M. DE BOER

You have probably run into this before: in program A you want to call a program B. The problem can arise that both programs use (partly) the same zero page locations. Or you use zero page locations that are also used by the monitor, which makes debugging impossible. The only solution is then to change all zero page addresses as desired. When you work with an assembler, that is a piece of cake. When the conversion has to be done by hand, it is tedious and time-consuming work with a large chance of errors. The "zero page shifter" does it for you! In a fraction of a second it selects the opcodes that refer to a zero page address, and then replaces the corresponding address (if desired).

Usage

Of course, before you start the program you have to enter some data about the program to be modified. First of all, the start address. The first two digits (high order) of this address go to $00E2, the last two digits (low order) to address $00E1. The end of the program to be modified must be marked with $FF. To change the zero page addresses, the program uses two tables. In the first table you indicate which addresses must be changed. This table runs from address 00D1 ... 00E0. You can change 16 addresses at a time. If fewer addresses have to be replaced, you must fill the rest of the table with '00'. In the second table (00C1 ... 00D0) you note, in the corresponding place, which address should take the place of the old address.

Example

As an example we take the subroutine 'LETTER', which we discussed in detail in the three preceding months. Suppose this subroutine is called by a main program that needs all zero page locations above $C4. We must move the zero page locations D0...D5 (which are used by 'LETTER') to an area where the main program is not bothered by this subroutine. In our example we use A8...AD for this, which are still free. First we enter the start address ($0600) by putting '00' at address $00E1, and '06' at address $00E2. Next we mark the end of the program by putting 'FF' at address $0883 (this is the first free location after the program). Finally we fill in the tables: from address 00D1 the addresses to be changed: D0, D1, D2, D3, D4, D5 and then 10 times '00' to fill up the table. From address 00C1 come the 'new' addresses, A8, A9, AA, AB, AC,

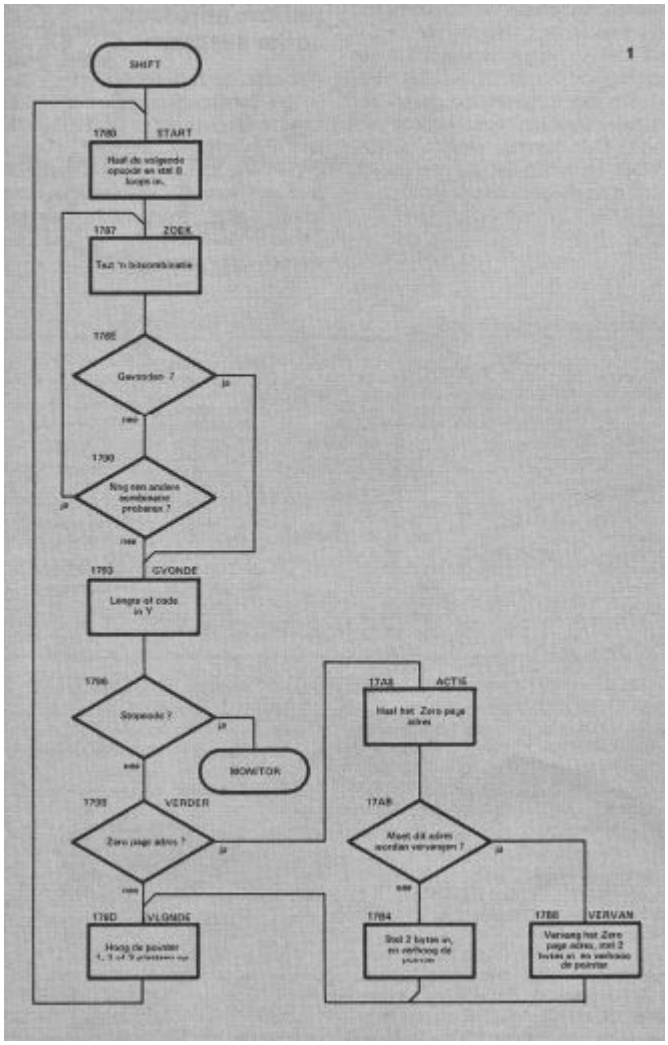
AD and again 10 times '00', to fill up this table as well. All data has now been entered, and the zero page shifter can be started at 1780. With short programs it will seem as if nothing happens; with somewhat longer programs the display goes dark for a short time. In any case you will see that the zero page shifter has performed its task flawlessly (his? her?).

Warning
After starting the zero page shifter, the pointer (00E1 and 00E2) will be at the end of the program to be modified. Therefore do not press 'GO' a second time, because the zero page shifter will then modify the memory space after the program, which can lead to undesired results. When a program is followed by a table, the stop code must be placed between the table and the program. It can also happen that a table in page 0 has to be moved. A table is often addressed with an address one lower than the start address of the table, e.g. LDA TAB-1,X. In this case the value TAB-1 must be given as the address to be changed.

Table 1

Register X	Opcode	Instructions	Length/code
8	XXXX XX11	stop code	FF
7	XXXX 11XX	abs and abs,X	03
6	XXX1 1001	abs,Y	03
5	0010 0000	JSR	03
4	XXXX 10X0	impl and accu	01
3	1X0 0000	RTI and RTS	01
2	XXXX 01XX	Zp, Zp,X and Zp,Y	00
1	XXXX 0001	(ind,X) and (ind),Y	00
00	remaining	relative, immediate	02

How it works
o run this program properly, the program has to know which memory locations contain an opcode and which contain an operand (address). For this it is necessary to determine the length of the instruction for each opcode. The principle of this length determination is more or less borrowed from the program 'relocate' (First Book of KIM). After the first opcode has been put in registers Y and A, the length determination begins. The opcode stays 'undamaged' in register Y during this process. The copy of the opcode in register A is repeatedly processed with data from the tables MASK and BITS. Index register X keeps track of which 'test' has been performed on the opcode. As an example we take X = 6. At address 1788 the opcode is thereby masked with the value $1F (the sixth value from MASK). So we look only at the last 5 bits of the opcode. At address 1788 the EOR operation is performed between the masked opcode and '0001 1001' (the sixth value from BITS). This means that bits 0, 3 and 4 are inverted. Only if this operation yields



$00 will the test be ended; otherwise we continue with X = 05. Which opcodes have we now selected? The first 3 bits were already set to 0 by the masking, so their initial value does not matter. All opcodes with the structure 'XXX1 1001' yield '00' in this test, so that with X = 6 we end up at address 1793. Here again the sixth value (now counting '0' as well) from the table is put into register Y. In this case that is 03, the length of the selected opcodes (in this case all ABS,Y instructions). In this way the program performs a maximum of 8 tests. If after the eighth test nothing has been found, we know for certain that the instruction is 2 bytes long, because then all 1- and 3-byte instructions, as well as part of the 2-byte instructions, have been filtered out. When a bit combination has been recognized, the length of the instruction is always put in register Y at address 1793. Only for the stop code does $FF end up in it, and for a zero page instruction $00 in register Y. This makes it possible to distinguish these 2 cases from the other opcodes. As long as nothing needs to be modified, at address 179D the pointer is incremented by one, two or three places, depending on the value in Y. This way it points exactly to the next opcode again, and the whole process starts over. At the stop code, as already said, register Y is loaded with FF, the only time a negative number is loaded. As a result, at address 1796 we do not jump over the 'JMP', and we return to the monitor. When the opcode was a zero page address, register Y is loaded with '00', so that at address 179B we can jump to 'ACTIE' with a BEQ. The rest of the program is not complicated: first the zero page address is copied into register A. Then, from address 17AB, it is checked whether this address occurs in the table of addresses to be changed. If so, the address is replaced. Before we continue with the next opcode, register Y must first be loaded with '02', to indicate that the pointer must be incremented by two places. A jump to 'VLGNDE' ($179D) then ensures that we start with the next opcode.

Other addresses, other system

When the zero page shifter has to work in a different memory area, the instructions at 1788, 178B and 1793 must be adjusted. On other systems the jump to the monitor (address 1798) must (also) be changed. Of course, this program can only be run by 65XX processors.

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 bitcombinatie
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 actie!
179D E6 E1    VLGNDE INC  POINTL     ┐
179F D0 02           BNE  $02        │
17A1 E6 E2           INC  POINTH     │ Hoog de pointer 1,2
17A3 88              DEY             ┘ of 3 plaatsen op
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     │
17AF F0 07           BEQ  VERVAN     │ Kijk of dit adres
17B1 CA              DEX             │ moet worden vervangen?
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 DF 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
```