Optimal Sort for any number of 16-bit elements

by Mats Rosengren

General Discussion

The extension of the "Optimal Sort" algorithm to "an unlimited number" (i.e. more then 255 elements) of 16 bit elements is straightforward using standard 6502 technique.

Consider first the loop using the Y register to sequentially access the elements starting with the second last and ending with the first in a sequence of up to 255 elements (note that the last access here is with 1 in the Y register, i.e. the "ZPADD" should point to the byte before the first element):

```
L1   DEY
     BEQ L3
     LDA (ZPADD),Y
     "some code"
     BRA L1
L3
```

To access a longer sequence one could use nested loops with the X register for the outer and the Y register for the inner loop as follows:

```
L1   TYA
     BNE L2
     TXA
     BEQ L3
     DEX
     DEC ZPADD+1
L2   DEY
     LDA (ZPADD),Y
     "some code"
     BRA L1
L3
```

If initially the registers X and Y are given the values x and y and the value y is added to high part of the pointer ZPADD pointing to the first element of the sequence to be sorted the loop will be run through y + 256 * x times starting with the second last and ending with the first in a sequence of 1 + y + 256 * x elements. And when the loop is finished the pointer ZPADD will again point to the first element of the sequence while the X and the Y registers will take the value zero. Note that as here the last access is with 0 in the Y register the "ZPADD" should indeed point to the first element, not to the byte before the first element

To work with 16 bit values one has to handle the 8 less significant bits and the 8 more significant bits

separately. The comparison is then made such that first the 8 most significant bits are compared. The 8 less significant bits only need to be compared if the 8 more significant bits were equal. The complete routine is then as follows:

```
SORT CLC
     LDA ZPADL+1        ;INITIALISE WORK4L
     ADC NVALH
     STA WORK4L
     BCS LER            ;BAD INPUT VALUES (TOO FAR TO BRANCH DIRECTLY)
     LDA ZPADH+1        ;INITIALISE WORK4H
     ADC NVALH
     STA WORK4H
LER  BCS L3B            ;BAD INPUT VALUES
L0   LDY NVALL          ;NUMBER OF DECREMENTS, LOW
     LDX NVALH          ;NUMBER OF DECREMENTS, HIGH
     LDA WORK4L
     STA ZPADL+1
     LDA (ZPADL),Y      ;LAST VALUE IN (WHAT IS LEFT OF) SEQUENCE TO BE
SORTED
     STA WORK3L         ;SAVE LEAST SIGNIFICANT PART OF VALUE. WILL BE OVER-
WRITTEN BY LARGEST NUMBER
     LDA WORK4H
     STA ZPADH+1
     LDA (ZPADH),Y      ;LAST VALUE IN (WHAT IS LEFT OF) SEQUENCE TO BE
SORTED
     STA WORK3H         ;SAVE MOST SIGNIFICANT PART OF VALUE. WILL BE OVER-
WRITTEN BY LARGEST NUMBER
     BRA L2
L1   TYA                ;START INNER LOOP
     BNE L1A
     TXA
     BEQ L3             ;EXIT INNER LOOP
     DEX
     DEC ZPADL+1
     DEC ZPADH+1
L1A  DEY
     LDA (ZPADH),Y
     CMP WORK2H
     BNE L1B
     LDA (ZPADL),Y
     CMP WORK2L
L1B  BCC L1
L2   LDA (ZPADL),Y
     STA WORK2L         ;POTENTIALLY LARGEST VALUE, LEAST SIGNIFICANT PART
     LDA (ZPADH),Y
     STA WORK2H         ;POTENTIALLY LARGEST VALUE, MOST SIGNIFICANT PART
     STY WORK1L         ;INDEX OF POTENTIALLY LARGEST VALUE, LOW
     LDA ZPADL+1
     STA WORK1LH        ;INDEX HIGH OF POTENTIALLY LARGEST VALUE, LEAST
SIGNIFICANT PART
     LDA ZPADH+1
```

```
     STA WORK1HH         ;INDEX HIGH OF POTENTIALLY LARGEST VALUE, MOST
SIGNIFICANT PART
     BRA L1              ;END INNER LOOP
L3   LDY NVALL           ;WHERE THE LARGEST VALUE SHALL BE PUT
     LDA WORK4L
     STA ZPADL+1
     LDA WORK2L          ;THE LARGEST VALUE, LEAST SIGNIFICANT PART
     STA (ZPADL),Y
     LDA WORK4H
     STA ZPADH+1
     LDA WORK2H          ;THE LARGEST VALUE, MOST SIGNIFICANT PART
     STA (ZPADH),Y
     LDY WORK1L          ;INDEX LOW OF FREE SPACE
     LDA WORK1LH         ;INDEX HIGH OF FREE SPACE, LEAST SIGNIFICANT PART
     STA ZPADL+1
     LDA WORK3L          ;THE OVER-WRITTEN VALUE, LEAST SIGNIFICANT PART
     STA (ZPADL),Y       ;PUT THE OVER-WRITTEN VALUE, LEAST SIGNIFICANT PART,
IN THE FREE SPACE
     LDA WORK1HH         ;INDEX HIGH OF FREE SPACE, MOST SIGNIFICANT PART
     STA ZPADH+1
     LDA WORK3H          ;THE OVER-WRITTEN VALUE, MOST SIGNIFICANT PART
     STA (ZPADH),Y       ;PUT THE OVER-WRITTEN VALUE, MOST SIGNIFICANT PART,
IN THE FREE SPACE
     LDA NVALL
     BNE L3A
     LDA NVALH
     BEQ L3B             ;EXIT OUTER LOOP
     DEC NVALH
     DEC WORK4L
     DEC WORK4H
L3A  DEC NVALL           ;END OF THE SHORTER SEQUENCE STILL LEFT
     BRA L0              ;START WORKING WITH THE SHORTER SEQUENCE
L3B  RTS
```

The calling program has to set the pointers ZPADL and ZPADH pointing to the start of the (separately stored) sequences of the 8 less and the 8 more significant bits of the values. The total number of values are 1 + y + 256 * x where the value x has to be given to variable NVALL and the value y has to be given to variable NVALH. At the end these variables will both be zero.

In addition the following 9 bytes are needed internally

```
WORK1L
WORK1LH
WORK1HH
WORK2L
WORK2H
WORK3L
WORK3H
WORK4L
WORK4H
```

The following is a template for a program calling the routine:

```
;
ZPADL   = $30           ;2 BYTE POINTER IN PAGE ZERO TO THE "LESS
SIGNIFICANT" SEQUENCE. SET BY CALLING PROGRAM
ZPADH   = $32           ;2 BYTE POINTER IN PAGE ZERO TO THE "MORE
SIGNIFICANT" SEQUENCE. SET BY CALLING PROGRAM
NVALL   = $34           ;< "START ADDRESS" - "END ADDRESS" OF SEQUENCE. SET
BY CALLING PROGRAM
NVALH   = $35           ;> "START ADDRESS" - "END ADDRESS" OF SEQUENCE. SET
BY CALLING PROGRAM
;
;9 BYTES USED AS WORKING AREA
;
WORK1L  = $36           ;LOW INDEX (Y) FOR POTENTIALLY LARGEST VALUE
WORK1LH = $37           ;HIGH INDEX (ADDL+1) FOR POTENTIALLY LARGEST VALUE
(LEAST SIGNIFICANT PART)
WORK1HH = $38           ;HIGH INDEX (ADDL+1) FOR POTENTIALLY LARGEST VALUE
(MOST SIGNIFICANT PART)
WORK2L  = $39           ;POTENTIALLY LARGEST VALUE (LEAST SIGNIFICANT PART)
WORK2H  = $3A           ;POTENTIALLY LARGEST VALUE (MOST SIGNIFICANT PART)
WORK3L  = $3B           ;COPY OF LAST VALUE IN NOT YET SORTED SEQUENCE
(LEAST SIGNIFICANT PART)
WORK3H  = $3C           ;COPY OF LAST VALUE IN NOT YET SORTED SEQUENCE (MOST
SIGNIFICANT PART)
WORK4L  = $3D           ;HIGH ADDRESS FOR LAST ELEMENT IN NOT YET SORTED
SEQUENCE (LEAST SIGNIFICANT PART)
WORK4H  = $3E           ;HIGH ADDRESS FOR LAST ELEMENT IN NOT YET SORTED
SEQUENCE (MOST SIGNIFICANT PART)
     *=$1000            ;CODE ANYWHERE IN RAM OR ROM
     LDA #< SEQL
     STA ZPADL
     LDA #> SEQL
     STA ZPADL+1
     LDA #< SEQH
     STA ZPADH
     LDA #> SEQH
     STA ZPADH+1
     LDA #$09
     STA NVALL
     LDA #$00
     STA NVALH
     JSR SORT
     BRK
SEQL .BYTE    $30
     .BYTE    $20
     .BYTE    $DF
     .BYTE    $5A
     .BYTE    $81
     .BYTE    $C7
     .BYTE    $62
```

```
     .BYTE    $6D
     .BYTE    $6A
     .BYTE    $76
SEQH .BYTE    $05
     .BYTE    $1F
     .BYTE    $16
     .BYTE    $0D
     .BYTE    $1E
     .BYTE    $17
     .BYTE    $18
     .BYTE    $10
     .BYTE    $26
     .BYTE    $10
"Insert the code for subroutine SORT here"
```

To test the efficiency (and correctness) of this algorithm I generated a totally random permutation of the numbers 0 - 9999 using a random number generator. This sequence was successfully sorted using subroutine SORT above. I measured the number of T1H counts of the W65C22S timer 1 during this sorting, it was 4324613. As T1H decrements every 256 PHI2 pulses and my W6502 runs at 8 megahertz this corresponds to 138.4 seconds.