Improvement Upon a Division Program by Leventhal

by W.D. Maurer

W. D. Maurer, George Washington University, S.E.A.S., Washington, DC 20052.

The purpose of this paper is to give an improvement in timing, without affecting memory requirements and by changing only two instructions, in a 16-bit-by-8-bit division program for the 6502, due to Leventhal (Ref. 1, pp. 8-15).

The original algorithm is reproduced in Figure 1. It assumes that the 16-bit number in hexadecimal locations 40 and 41 (with bytes reversed, as usual) is being divided by the 8-bit number in location 42 to produce a quotient in location 43 and a remainder in location 44. All quantities are unsigned, but the leftmost bit of location 41 and the leftmost bit of location 42 are both assumed to be zero; this would be the case if the division were originally a signed division and the absolute values of the signed quantities had been taken. Overflow and division by zero are not checked.

The improvement is concerned with eliminating the INC instruction. Notice that INC adds one to the quotient, thus setting the current bit of the quotient to 1. If the INC is not done, the current bit of the quotient is zero.

Notice, however, that if INC is executed, SBC has just been executed, and this subtraction has produced a nonnegative result. Hence the carry bit will be equal to 1. On the other hand, if INC is skipped, by branching on carry clear to CHCNT, then the carry bit will be equal to zero.

Hence, in either case, the carry bit will be equal to the current bit of the quotient. This will be true at CHCNT, and therefore also at DIVID when we go back to the start of the loop, so that the bit can be rotated into the quotient by changing the ASL $43 to ROL $43 at this point.

The last time through the loop, the final bit of the quotient will still be in the carry bit, so that an extra ROL $43 is necessary immediately following the BNE. The two instructions changed are therefore:

(1) Change ASL $43 to ROL $43.
(2) Eliminate INC $43 and put another ROL $43 between BNE DIVID and STA $44. (All of the four instructions above have the same length, so that the memory requirements of the program are not affected.)

Note that the *first* time through the loop, the ROL at DIVID inserts a nonsense bit into hexadecimal location 43. This nonsense bit is then rotated all the way through this location and becomes the final value of the carry bit when the

Figure 1

```
                LDX   #8        ;NUMBER OF BITS IN DIVISOR = 8
                LDA   $40       ;START WITH LSB'S OF DIVIDEND
                STA   $43
                LDA   $41       ;GET MSB'S OF DIVIDEND
        DIVID   ASL   $43       ;SHIFT DIVIDEND, QUOTIENT LEFT 1 BIT
                ROL   A
                CMP   $42       ;CAN DIVISOR BE SUBTRACTED?
                BCC   CHCNT     ;NO, GO TO NEXT STEP
                SBC   $42       ;YES, SUBTRACT DIVISOR (CARRY = 1)
                INC   $43       ;AND INCREMENT QUOTIENT BY 1
        CHCNT   DEX             ;LOOP UNTIL ALL 8 BITS HANDLED
                BNE   DIVID
                STA   $44       ;STORE REMAINDER
                BRK
```

final ROL is done.

This fact suggests a further improvement in the program. We have noted that overflow and division by zero are not checked in this program and therefore must be checked externally; this takes at least two instructions. Suppose, however, that we were to check this in the given division routine itself. These two instructions would then be moved into the routine from outside it, thus causing again, no change in space requirements. The instructions might be CMP $42 and BCS ERROR inserted just before DIVID. (This works because the contents of location 41 are in the A register at this point. If the contents of location 42 are less than or equal to this, the carry will be set, and we have either the division-by-zero case, where location 42 contains zero, or the overflow case.)

Now notice that, in this case, the nonsense bit, as mentioned above, will always be zero, because when the first ROL is done the carry will be clear (otherwise we branched on carry set). Therefore the final value of the carry bit will also be zero, after the nonsense bit has passed through location 43. On the other hand, in either of the two abnormal cases, the carry will be set.

The further improvement is now as follows. Instead of branching to ERROR on carry set, we simply branch to the end of the program. We can now make the program into a subroutine by adding an RTS, and since this leaves the carry flag unaffected, it can serve, in the *calling* program, as an indication of error. Thus, for example, if the name of the subroutine is DIV, then JSR DIV followed by BCS ERROR will serve, in the calling program, to go to ERROR in either of the error cases and continue in the normal case. The program with all these improvements is illustrated in Figure 2.

If only the initial two changes are made, the timing improvement will be $5(n-1)$ cycles, where n is the number of one-bits in the quotient. The first change makes no difference; the new ROL takes 5 cycles, outside the loop, while the eliminated INC takes 5 cycles for each one-bit in the quotient.

DDJ

Reference

[1] Leventhal, L. A., *6502 Assembly Language Programming*, Osborne/McGraw-Hill, Berkeley, California.

Figure 2

```
DIV     LDX   #8      ;NUMBER OF BITS IN DIVISOR = 8
        LDA   $40     ;START WITH LSB'S OF DIVIDEND
        STA   $43
        LDA   $41     ;GET MSB'S OF DIVIDEND
        CMP   $42     ;SHOULD BE LESS THAN DIVISOR
        BCS   DONE    ;IF NOT, ERROR EXIT (CARRY = 1)
DIVID   ROL   $43     ;SHIFT DIVIDEND, QUOTIENT LEFT 1 BIT
        ROL   A       ;(AND NEW ANSWER BIT -- SEE DEX BELOW)
        CMP   $42     ;CAN DIVISOR BE SUBTRACTED?
        BCC   CHCNT   ;NO, GO TO NEXT STEP (CARRY = 0)
        SBC   $42     ;YES, SUBTRACT DIVISOR (CARRY = 1)
CHCNT   DEX           ;NOTE CARRY = NEW ANSWER BIT
        BNE   DIVID   ;LOOP UNTIL ALL 8 BITS HANDLED
        ROL   $43     ;SHIFT IN THE LAST ANSWER BIT
        STA   $44     ;STORE REMAINDER (CARRY = 0 HERE)
DONE    RTS           ;QUIT (CARRY 0, NORMAL, CARRY 1, ERROR)
```