Hexadecimal to Decimal Conversion

Without division or multiplication, this first routine takes an 8-bit hex number and returns a 10-bit decimal (BCD) number. It is done by the “add three” algorithm from Motorola AN-757, “Analog-to-Digital Conversion Techniques With the 6800 Microprocessor System” by Don Aldridge. Motorola apparently does not have the app. note on their website. I converted it in the 80's for a 65c02 product. Start with the input number in accumulator.

```
; A       = Hex input number (gets put into HTD_IN)
; HTD_OUT   = 1s & 10s output byte
; HTD_OUT+1 = 100s output byte

HTD:    CLD             ; (Make sure it's not in decimal mode for the
        STA HTD_IN      ;                ADCs below.)
        TAY             ; Save the input to restore later if desired.
        STZ HTD_OUT+1   ; Begin by storing 0 in the output bytes.
        STZ HTD_OUT     ; (NMOS 6502 will need LDA #0, STA ...)
        LDX #8

 htd1$: ASL HTD_IN
        ROL HTD_OUT
        ROL HTD_OUT+1

        DEX             ; The shifting will happen seven times.  After
        BEQ htd3$       ; the last shift, you don't check for digits of
                        ; 5 or more.
        LDA HTD_OUT
        AND #FH
        CMP #5
        BMI htd2$

        CLC
        LDA HTD_OUT
        ADC #3
        STA HTD_OUT

 htd2$: LDA HTD_OUT
        CMP #50H
        BMI htd1$

        CLC
        ADC #30H
        STA HTD_OUT

        BRA htd1$       ; NMOS 6502 can use JMP.

 htd3$: STY HTD_IN      ; Restore the original input.
        RTS
```

The above method is interesting and I used it in a product; but with a little thought I should have applied myself to years ago, I quickly saw there's a simpler and more efficient way to do the same thing:

```
HTD_IN:     BLKB    1
HTD_OUT:    BLKB    2       ; low byte first

TABLE:      WORD    1, 2, 4, 8, 16H, 32H, 64H, 128H
                ; (Word directive puts low byte first.)

HTD:    SED             ; Output gets added up in decimal.
        STZ HTD_OUT     ; Inititalize output word as 0.
        STZ HTD_OUT+1   ; (NMOS 6502 will need LDA#0, STA ...)

        LDX #0EH        ; $E is 14 for 2x7 bits.  (0-7 is 8 positions.)
 loop:  ASL HTD_IN      ; Look at next high bit.  If it's 0,
        BCC htd1$       ; don't add anything to the output for this bit.
        LDA HTD_OUT     ; Otherwise get the running output sum
        CLC
        ADC TABLE,X     ; and add the appropriate value for this bit
        STA HTD_OUT     ; from the table, and store the new sum.
        LDA HTD_OUT+1   ; After low byte, do high byte.
        ADC TABLE+1,X
        STA HTD_OUT+1

 htd1$: DEX             ; Go down to next bit value to loop again.
        DEX
        BPL loop        ; If still not done, go back for another loop.

        CLD
        RTS
```

The principle should be pretty clear. You can take it out to as many digits as you want. Here it is for 16-bit hex input to 5-digit decimal output. Since the entire input number may not fit in A, put it in HTD_IN first.

```
HTD_IN:     BLKB    2    ; Low byte first, as is normal for 6502.
HTD_OUT:    BLKB    3   ; Low byte first, highest byte last.

                ; The table below has high byte first just to
                ; make it easier to see the number progression.
TABLE:  BYTE    0, 0H, 1H,  0, 0H, 2H,  0, 0H, 4H,  0, 0H, 8H
        BYTE    0, 0H,16H,  0, 0H,32H,  0, 0H,64H,  0, 1H,28H
        BYTE    0, 2H,56H,  0, 5H,12H,  0,10H,24H,  0,20H,48H
        BYTE    0,40H,96H,  0,81H,92H,  1,63H,84H,  3,27H,68H

HTD:    SED             ; Output gets added up in decimal.

        STZ HTD_OUT     ; Inititalize output as 0.
        STZ HTD_OUT+1   ; (NMOS 6502 will need LDA#0, STA...)
```

```
        STZ HTD_OUT+2

        LDX #2DH        ; 2DH is 45 decimal, or 3x15 bits.
loop:   ASL HTD_IN      ; (0 to 15 is 16 bit positions.)
        ROL HTD_IN+1    ; If the next highest bit was 0,
        BCC htd1$       ; then skip to the next bit after that.
        LDA HTD_OUT     ; But if the bit was 1,
        CLC             ; get ready to
        ADC TABLE+2,X   ; add the bit value in the table to the
        STA HTD_OUT     ; output sum in decimal--  first low byte,
        LDA HTD_OUT+1   ; then middle byte,
        ADC TABLE+1,X
        STA HTD_OUT+1
        LDA HTD_OUT+2   ; then high byte,
        ADC TABLE,X     ; storing each byte
        STA HTD_OUT+2   ; of the summed output in HTD_OUT.

htd1$:  DEX             ; By taking X in steps of 3, we don't have to
        DEX             ; multiply by 3 to get the right bytes from the
        DEX             ; table.
        BPL loop

        CLD
        RTS
```

Note by Karoshier: The above code can be easily made a little faster and shorter by splitting the data table into three separate tables, one for the low digits, one for the mid digits and one for the high digits. This way the index can be initialized to 15 instead of 45 and be decremented by 1 instead of 3, thereby saving 4 cycles per iteration and making the routine shorter by 2 bytes. If you have space constraints and you need an 8 bit version as well, you can also make the 16 bit version work for both 8 or 16 bits by changing the initial value of X according to the needs (namely 3 times 7 instead of 3 times 15) and patching the ROL opcode on the fly to a BIT instruction, which does not alter the carry.

See also More Hexadecimal to Decimal Conversion. Andrew Jacobs has written equivalent routines to the ones presented here, eliminating the lookup tables in exchange for a little slower execution speed.