BCD and Decimal Adjust

The decimal adjust operation on the Z80 is often confusing. In order to understand decimal adjust, it is necessary to understand binary coded decimal.

The idea behind binary coded decimal (BCD) is to store two decimal digits in a byte. For example, to store 42 in a byte, the BCD encoding is 4 in the upper four bits and 2 in the lower 4 bits, i.e. binary 0100 0010 or hex 0x42.

Note that only the interpretation of the byte changes. Instead of treating the byte as binary corresponding to the decimal value 66, it is treated as BCD corresponding to the decimal value 42.

The motivation behind BCD is that computer programs often receive decimal input and produce decimal output. By storing the values in decimal inside the computer, costly conversions between binary and decimal are avoided.

Converting between binary and decimal requires multiplying and dividing by 10. This is trivial with modern computers, but the Z80 doesn't have hardware multiply and divide. Instead, multiplication and division must be implemented as loops, which are relatively slow. In addition, the extra code may be impractical when memory is limited.

One drawback of BCD is that storage is less efficient since a byte only holds 100 values rather than 256 values.

A drawback of BCD is that standard binary arithmetic doesn't quite work. If you add BCD values 42 and 25 using standard addition, the result is 0x42 + 0x25 = 0x67 corresponding to BCD 67, and everything works fine. However, if you add BCD values 42 and 9 (stored as 0x42 and 0x09), the sum is 0x4b which isn't valid BCD (it would be something like 4 11). Another problem is illustrated by adding BCD values 19 and 19. Using binary addition, this is computed as 0x19 + 0x19 = 0x32, which would be interpreted as BCD 32. While both digits are valid, the answer is wrong, since it should be BCD 38.

The problem occurs because BCD values are being added with a binary adder. One solution would be to implement a separate BCD adder in the ALU, but this would be costly in terms of hardware. Instead the Z80 provides a separate DAA (Decimal Adjust Accumulator) instruction. By executing this instruction after an addition, subtraction, or similar operation, the result is corrected to a BCD value. In the above cases, the sum 0x4b would be corrected to 0x51 (BCD 51), and the sum 0x32 would be corrected to 0x38 (BCD 38).

To correct the sums above, it was necessary to add 0x06 to the sum. In general, to correct a sum, the DAA operation adds 0x00, 0x06, 0x60, or 0x66 to the value. Specifically, if there is a carry out of the low digit, or there should have been a carry (the low-digit sum is greater than 9), then 0x06 is added. Similarly, if there is a carry out of the high digit, or there should have been a carry, then 0x60 is added to the high digit. If both digits need correcting, then 0x66 is added. If neither digit needs correcting then nothing is added.

The reason for this correction factor is that a decimal carry happens at 10, but a binary carry out of 4 bits happens at 16. To fix the BCD value, the “missing” 6 counts are added.

Subtraction is similar, but in the opposite direction. For instance, BCD 22 minus BCD 14 is computed as 0x22-0x14=0x0e, which isn't a valid BCD value. Subtracting 0x06 yields 0x08 (BCD 08), which is the correct value.

Instead of subtracting 0x00, 0x06, 0x60, or 0x66, the Z80 does a decimal adjust after subtraction by adding the two's-complement value: 0x00, 0xfa, 0xa0, or 0x9a.

The N flag bit is used to keep track if the previous instruction was an add (N=0) or subtract (N=1).

Several circuits are used to implement DAA. Some logic tests if the low-order 4 bits latched in the ALU are less than 10. Additional logic tests if the high-order 4 bits in the ALU are less than 10 (with an implied carry if the low-order bits are 10 or greater). The first value, along with the H flag, determine if 6 should be added to the low-order 4 bits. The second value, along with the C flag, determine if 6 should be added to the high-order 4 bits. (If so, the carry flag is set.) The resulting constant is fed onto the regbus, where it is used by the accumulator. The N flag selects if the value is complemented or not.

To summarize, the low-order digit requires correction if:
a) the current value is 10 or greater
or, b) the half-carry flag is set, indicating a carry out of the digit for addition, or a borrow for the digit for subtraction.

The high-order digit requires correction if:
a) the current value is 10 or greater
a') the current value is 9 and the low-order digit is 10 or greater
or, b) the carry flag is set, indicating a carry for addition, or a borrow for subtraction.