How DP instructions affect Flags C, V, Z, N

<table>
<tr><td rowspan="2">Instructions</td><td colspan="3">Effect on Flags</td></tr>
<tr><td>if S-bit = 0</td><td>if S-bit = 1 and
no shift/rotate</td><td>if S-bit = 1 and
shift/rotate is there</td></tr>
<tr><td>add, sub, rsb,
adc, sbc, rsc</td><td>No flags are affected</td><td colspan="2">All 4 flags are affected, ALU carry is used</td></tr>
<tr><td>cmp, cmn</td><td colspan="3">All 4 flags are affected, ALU carry is used</td></tr>
<tr><td>and, orr, xor, bic,
mov, mvn</td><td>No flags are affected</td><td>Only Z and N are
affected</td><td rowspan="2">C, Z and N are affected
shift/rotate carry is used</td></tr>
<tr><td>tst, teq</td><td colspan="2">Only Z and N are affected</td></tr>
</table>

Circuit for maintaining Flags

This circuit will, of course, have an edge triggered 4-bit register (or 4 edge triggered flip-flops) that will be set according to the table shown above. The inputs required for it are as follows.

- Whether it is a DP instruction? If yes, which of the 16 instructions it is or to which of the 4 sub-classes of DP instructions (as per the rows of the above table) it belongs?
- S-bit of the instruction.
- Whether shift/rotate for operand 2 is specified or not? No shift/rotate is encoded as LSL #0. Other three types of shifts with #0 (i.e., LSR #0, ASR #0 and ROR #0) are used for some special purpose and are out of scope for the current Lab assignment.
- Carry output from ALU.
- Carry output from shifter.
- Result output from ALU (for determining next value of Z and N flags).
- MSB's of ALU operands (for determining next value of V flag).

Determining the next value of V flag

Let us use the following notation.

A_{31} and B_{31} are the msb's of the ALU operands
S_{31} is the msb of the result produced by ALU
C_{32} and C_{31} are the final and per-final carries from ALU

If both C_{32} and C_{31} are available, the simplest way to determine overflow is as follows.

$V = C_{31} \text{ xor } C_{32}$ [1]

However, since we are using + operator of VHDL rather than doing bit-by-bit addition, C_{31} is not available to us. We can determine overflow by looking at the signs of operands and result, as shown below (assuming add operation being performed by ALU).

$V = A_{31}.B_{31}.S_{31}' + A_{31}'.B_{31}'.S_{31}$ [2]

Alternatively, we can first determine C_{31} as shown below and then apply equation [1].

Since $S_{31} = A_{31}$ xor B_{31} xor C_{31}, we get $C_{31} = A_{31}$ xor B_{31} xor S_{31} [3]

Note that while equation [1] does not depend on whether the operation being performed by ALU is addition or subtraction, equations [2] and [3] assume the operation to be add operation. In case of subtract operation, we need to replace B_{31} by B_{31}' and in case of reverse subtract operation, we need to replace A_{31} by A_{31}'.