“Computers in the future may weigh no more than 1.5 tons” Popular Science, 1949

Recap
Debugging: Monitor, dump
TExaS
Real 9S12DG128

Overview
Addition and subtraction set CCR bits
Subtraction used for conditional branching

Read sections 3.8, and 5.2 in the book
Watch movies on Example 3.9, 3.10, 3.11, 3.12 on the web

Condition code register (CC or CCR)
C set after an unsigned add if the answer is wrong
V set a signed add if the answer is wrong

<table>
<thead>
<tr>
<th>bit</th>
<th>name</th>
<th>meaning after add or sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>negative</td>
<td>result is negative</td>
</tr>
<tr>
<td>Z</td>
<td>zero</td>
<td>result is zero</td>
</tr>
<tr>
<td>V</td>
<td>overflow</td>
<td>signed overflow</td>
</tr>
<tr>
<td>C</td>
<td>carry</td>
<td>unsigned overflow</td>
</tr>
</tbody>
</table>

Table 3.16. Condition code bits.

Observation: The carry bit, C, is set after an unsigned addition or subtraction when the result is incorrect.
Observation: The overflow bit, V, is set after a signed addition or subtraction when the result is incorrect.

Let the result R be the result of the addition A+B.

N bit is set
- if unsigned result is above 127 or
- if signed result is negative.

\[ N = R_7 \]

Z bit is set if result is zero.

\[ Z = \overline{R_7} & \overline{R_6} & \overline{R_5} & \overline{R_4} & \overline{R_3} & \overline{R_2} & \overline{R_1} & \overline{R_0} \]

V bit is set after a signed addition if result is incorrect

\[ V = A_7 & B_7 & \overline{R_7} + A_7 & B_7 & R_7 \]

C bit is set after an unsigned addition if result is incorrect

\[ C = A_7 & B_7 + A_7 & \overline{R_7} + B_7 & \overline{R_7} \]

Let the result R be the result of the subtraction A-B.

N bit is set
- if unsigned result is above 127 or
- if signed result is negative.

\[ N = R_7 \]

Z bit is set if result is zero.
\[ Z = R_7 \& R_6 \& R_5 \& R_4 \& R_3 \& R_2 \& R_1 \& R_0 \]

V bit is set after a signed subtraction if result is incorrect
\[ V = A_7 \& B_7 \& R_7 + A_7 \& B_7 \& R_7 \]

C bit is set after an unsigned subtraction if result is incorrect
\[ C = A_7 \& B_7 + R_7 + A_7 \& R_7 \]

**Question 1a.** What will be the value of the overflow (V) bit after executing the following?
```
ldaa #-100
adda #50
```

**Question 1b.** What will be the value of the carry (C) bit after executing the following?
```
ldaa #156
adda #50
```

**Question 2a.** What will be the value of the overflow (V) bit after executing the following?
```
ldaa # -100
adda # -50
```

**Question 2b.** What will be the value of the carry (C) bit after executing the following?
```
ldaa # 156
adda # 206
```

**Question 3.** What will be the value of the carry (C) bit after executing the following?
```
ldab #210
subb #60
```

**Question 4.** What will be the value of the overflow (V) bit after executing the following?
```
ldaa #-70
suba #-60
```

*Common Error:* Ignoring overflow (signed or unsigned) can result in significant errors.

*Observation:* Microcomputers have two sets of conditional branch instructions (if statements) that make program decisions based on either the C or V bit.

Promotion involves increasing the precision of the input numbers, and performing the operation at that higher precision.

<table>
<thead>
<tr>
<th>decimal</th>
<th>8-bit</th>
<th>16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>224</td>
<td>1110,0000</td>
<td>0000,0000,1110,0000</td>
</tr>
<tr>
<td>+ 64</td>
<td>+0100,0000</td>
<td>+0000,0000,0100,0000</td>
</tr>
<tr>
<td>288</td>
<td>0010,0000</td>
<td>0000,0001,0010,0000</td>
</tr>
</tbody>
</table>

We can check the 16-bit intermediate result to see if the answer will fit back into the 8-bit result.

![Figure 3.25. Promotion to detect and correct unsigned arithmetic errors.](image-url)
Write C code to solve one of these

To promote a signed number, we duplicate the sign bit

<table>
<thead>
<tr>
<th>decimal</th>
<th>8-bit</th>
<th>16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>-96</td>
<td>1010,0000</td>
<td>1111,1111,1010,0000</td>
</tr>
<tr>
<td>-64</td>
<td>0100,0000</td>
<td>-0000,0000,0100,0000</td>
</tr>
<tr>
<td>-160</td>
<td>0110,0000</td>
<td>1111,1111,0110,0000</td>
</tr>
</tbody>
</table>

```c

[bcc l1] ;jump to l1 if C=0
[bc s l2] ;jump to l2 if C=1
[bvc l3] ;jump to l3 if V=0
[bvs l4] ;jump to l4 if V=1
[bpl l5] ;jump to l5 if N=0
[bmi l6] ;jump to l6 if N=1
[bne l7] ;jump to l7 if Z=0
[beq l8] ;jump to l8 if Z=1
```

Figure 3.26. Flowcharts showing how to use promotion to detect and correct signed arithmetic errors.

Write C code to solve one of these

**Common Error:** Even though most C compilers automatically promote to a higher precision during the intermediate calculations, they do not check for overflow when demoting the result back to the original format.

```c

[bcc l1] ;jump to l1 if C=0
[bcs l2] ;jump to l2 if C=1
[bvc l3] ;jump to l3 if V=0
[bvs l4] ;jump to l4 if V=1
[bpl l5] ;jump to l5 if N=0
[bmi l6] ;jump to l6 if N=1
[bne l7] ;jump to l7 if Z=0
[beq l8] ;jump to l8 if Z=1
```

**ceiling and floor**

```c

[bcc l1] ;jump to l1 if C=0
[bcs l2] ;jump to l2 if C=1
[bvc l3] ;jump to l3 if V=0
[bvs l4] ;jump to l4 if V=1
[bpl l5] ;jump to l5 if N=0
[bmi l6] ;jump to l6 if N=1
[bne l7] ;jump to l7 if Z=0
[beq l8] ;jump to l8 if Z=1
```

Figure 3.27. Flowcharts showing how to use overflow bits to detect and correct unsigned arithmetic errors.

Assume A8, B8 and R8 are three 8-bit (1-byte) global variables defined in RAM.

```c

A8 ds 1 ;Input
B8 ds 1 ;Input
R8 ds 1 ;Output
```

The following assembly language adds two unsigned 8-bit numbers, using the algorithm presented in Figure 2.33.

```assembly

ldaa A8 ;get first input
adda B8 ;A8+B8
```

Jonathan W. Valvano
bcc OK1 ;if C=0, then no error,  
ldaa #255 ;overflow  
OK1 staa R8

The following assembly language subtracts two unsigned 8-bit numbers.

ldaa A8 ;get first parameter  
suba B8 ;A8-B8  
bcc OK2 ;if C=0, then no error,  
ldaa #0 ;underflow  
OK2 staa R8

<table>
<thead>
<tr>
<th>C code</th>
<th>assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(G2 == G1){</td>
<td>ldaa G2</td>
</tr>
<tr>
<td>isEqual();</td>
<td>cmpa G1</td>
</tr>
<tr>
<td>}</td>
<td>bne next jsr isEqual next</td>
</tr>
<tr>
<td>if(G2 != G1){</td>
<td>ldaa G2</td>
</tr>
<tr>
<td>isNotEqual();</td>
<td>cmpa G1</td>
</tr>
<tr>
<td>}</td>
<td>beq next jsr isNotEqual next</td>
</tr>
<tr>
<td>if(H2 == H1){</td>
<td>ld d H2</td>
</tr>
<tr>
<td>isEqual();</td>
<td>cpd H1</td>
</tr>
<tr>
<td>}</td>
<td>bne next jsr isEqual next</td>
</tr>
<tr>
<td>if(H2 != H1){</td>
<td>ld d H2</td>
</tr>
<tr>
<td>isNotEqual();</td>
<td>cpd H1</td>
</tr>
<tr>
<td>}</td>
<td>beq next jsr isNotEqual next</td>
</tr>
</tbody>
</table>

**Table 5.1. Conditional structures that test for equality.**

**Signed conditional branch**

<table>
<thead>
<tr>
<th>C code</th>
<th>assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(G2 &gt; G1){</td>
<td>ldaa G2</td>
</tr>
<tr>
<td>isGreater();</td>
<td>cmpa G1</td>
</tr>
<tr>
<td>}</td>
<td>ble next jsr isGreater next</td>
</tr>
<tr>
<td>if(G2 &gt;= G1){</td>
<td>ldaa G2</td>
</tr>
<tr>
<td>isGreaterEq();</td>
<td>cmpa G1</td>
</tr>
<tr>
<td>}</td>
<td>blt next jsr isGreaterEq next</td>
</tr>
</tbody>
</table>
Table 5.3. Signed conditional structures.

Unsigned conditional branch

<table>
<thead>
<tr>
<th>Assembly code</th>
<th>C Code</th>
</tr>
</thead>
</table>
| bhs target ; Branch if unsigned greater than or equal to, | if(G2 > G1){
| ;if C=0, same as bcc | isGreater();
| | }
| bhi target ; Branch if unsigned greater than, | if(G2 >= G1){
| ;if C+Z=0 | isGreaterEq();
| blo target ; Branch if unsigned less than, | if(G2 < G1){
| ;if C=1, same as bcs | isLess();}
| bls target ; Branch if unsigned less than or equal to, | if(G2 <= G1){
| ;if C+Z=1 | isLessEq();}

Table 5.2. Unsigned conditional structures.

The bottom line

Use C bit, bhi, bhs, blo, bls for unsigned numbers
Use V bit, bgt, bge, blt, ble for signed numbers
Zero pad for unsigned 8 to 16 bit conversion
Sign extend for signed 8 to 16 bit conversion
Overflow detection using C and V bits
Overflow correction using promotion or ceiling/floor