"*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

bit	name	meaning after add or sub
N	negative	result is negative
Z	zero	result is zero
V	overflow	signed overflow
C	carry	unsigned overflow

Table 3.16. Condition code bits.

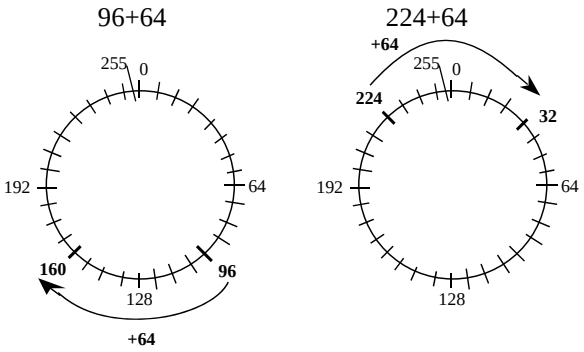

Figure 3.20. Unsigned number wheel.

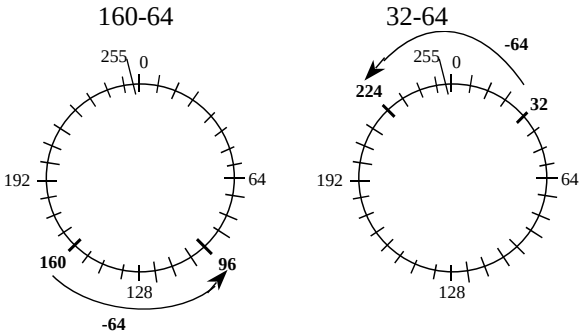

Figure 3.22. Unsigned number wheel.

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

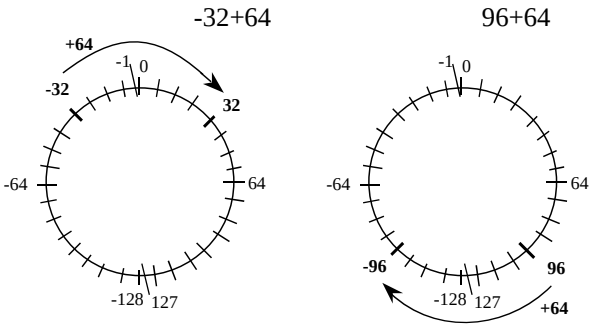

Figure 3.23. Signed number wheel.

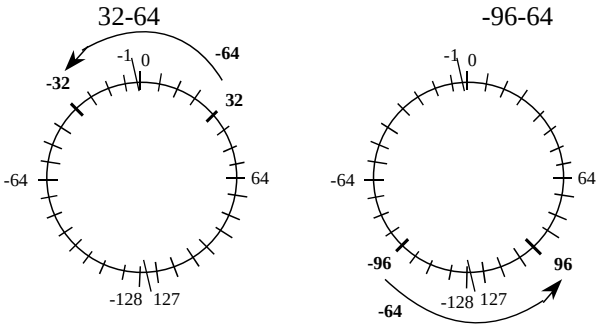

Figure 3.24. Signed number wheel

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 = R7

Z bit is set if result is zero.

$$Z = \overline{R7} \,\&\, \overline{R6} \,\&\, \overline{R5} \,\&\, \overline{R4} \,\&\, \overline{R3} \,\&\, \overline{R2} \,\&\, \overline{R1} \,\&\, \overline{R0}$$

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

$$V = A7 \,\&\, B7 \,\&\, \overline{R7} + \overline{A7} \,\&\, \overline{B7} \,\&\, R7$$

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

$$C = A7 \,\&\, B7 + A7 \,\&\, \overline{R7} + B7 \,\&\, \overline{R7}$$

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 = R7

Z bit is set if result is zero.

$$Z = \overline{R7} \,\&\, \overline{R6} \,\&\, \overline{R5} \,\&\, \overline{R4} \,\&\, \overline{R3} \,\&\, \overline{R2} \,\&\, \overline{R1} \,\&\, \overline{R0}$$

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

$$V = A7 \,\&\, \overline{B7} \,\&\, \overline{R7} + \overline{A7} \,\&\, B7 \,\&\, R7$$

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

$$C = \overline{A7} \,\&\, B7 + B7 \,\&\, R7 + \overline{A7} \,\&\, R7$$

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.

decimal	8-bit	16-bit
224	1110,0000	0000,0000,1110,0000
+ 64	+0100,0000	+0000,0000,0100,0000
288	0010,0000	0000,0001,0010,0000

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

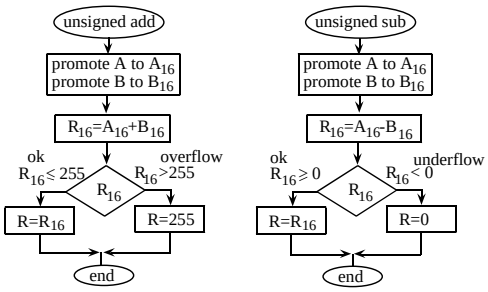

Figure 3.25. Promotion to detect and correct unsigned arithmetic errors.

Write C code to solve one of these

To promote a signed number, we duplicate the sign bit

```
decimal        8-bit              16-bit
  -96         1010,0000      1111,1111,1010,0000
- 64         -0100,0000     -0000,0000,0100,0000
-160          0110,0000      1111,1111,0110,0000
```

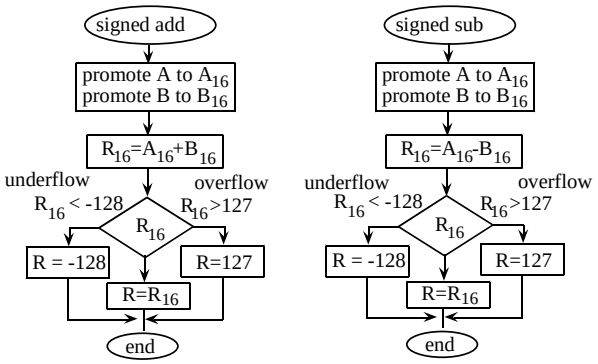

Figure 3.26. Flowcharts showing how to use promotion to detect and correct signed arithmetic errors.

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.*

```
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

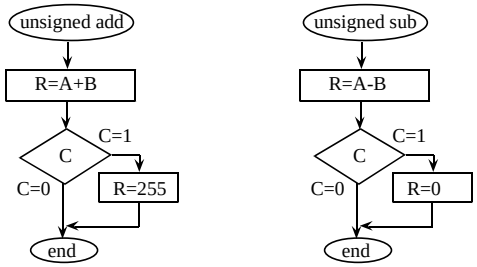

Figure 3.27. Flowcharts showing how to use overflow bits to detect and correct unsigned arithmetic errors.

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

```
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.

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

```
      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
```

C code	assembly code

`if(G2 ==  G1){`
`  isEqual();`
`}`
	`   ldaa G2`
	`   cmpa G1`
	`   bne  next`
	`   jsr  isEqual`
	`next`

`if(G2 != G1){`
`  isNotEqual();`
`}`
	`   ldaa G2`
	`   cmpa G1`
	`   beq  next`
	`   jsr  isNotEqual`
	`next`

`if(H2 == H1){`
`  isEqual();`
`}`
	`   ldd  H2`
	`   cpd  H1`
	`   bne  next`
	`   jsr  isEqual`
	`next`

`if(H2 != H1){`
`  isNotEqual();`
`}`
	`   ldd  H2`
	`   cpd  H1`
	`   beq  next`
	`   jsr  isNotEqual`
	`next`

Table 5.1. Conditional structures that test for equality.

Signed conditional branch

```
      bge target   ;Branch if signed greater than or equal to,
                         ;if (N^V)=0, or (~N•V+N•~V)=0
      bgt target   ;Branch if signed greater than,
                         ;if (Z+N^V)=0, or (Z+~N•V+N•~V)=0
      ble target   ;Branch if signed less than or equal to,
                         ;if (Z+N^V)=1, or (Z+~N•V+N•~V)=1
      blt target   ;Branch if signed less than,
                         ;if (N^V)=1, or (~N•V+N•~V)=1
```

C code	assembly code

`if(G2 > G1){`
`  isGreater();`
`}`
	`   ldaa G2`
	`   cmpa G1`
	`   ble  next`
	`   jsr  isGreater`
	`next`

`if(G2 >= G1){`
`  isGreaterEq();`
`}`
	`   ldaa G2`
	`   cmpa G1`
	`   blt  next`
	`   jsr  isGreaterEq`
	`next`

`if(G2 < G1){`
`  isLess();`
`}`
	`  ldaa G2`
	`  cmpa G1`
	`  bge  next`
	`  jsr  isLess`
	`next`

`if(G2 <= G1){`
`  isLessEq();`
`}`
	`  ldaa G2`
	`  cmpa G1`
	`  bgt  next`
	`  jsr  isLessEq`
	`next`

Table 5.3. Signed conditional structures.

Unsigned conditional branch

```
      bhs target   ;Branch if unsigned greater than or equal to,
                         ;if C=0, same as bcc
      bhi target   ;Branch if unsigned greater than,
                         ;if C+Z=0
      blo target   ;Branch if unsigned less than,
                         ;if C=1,  same as bcs
      bls target   ;Branch if unsigned less than or equal to,
                         ;if C+Z=1
```

C code	assembly code

`if(G2 > G1){`
`  isGreater();`
`}`
	`  ldaa G2`
	`  cmpa G1`
	`  bls  next`
	`  jsr  isGreater`
	`next`

`if(G2 >= G1){`
`  isGreaterEq();`
`}`
	`  ldaa G2`
	`  cmpa G1`
	`  blo  next`
	`  jsr  isGreaterEq`
	`next`

`if(G2 < G1){`
`  isLess();`
`}`
	`  ldaa G2`
	`  cmpa G1`
	`  bhs  next`
	`  jsr  isLess`
	`next`

`if(G2 <= G1){`
`  isLessEq();`
`}`
	`  ldaa G2`
	`  cmpa G1`
	`  bhi  next`
	`  jsr  isLessEq`
	`next`

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