Documentation Développeurs
Apple Computer France 1987

Document développeur numéro 40

Numéric Magnitude comp on IIGS

type d'upgrade de ce ducument : 4
1 Documentation de première catégorie inchangée
2 Documentation de deuxième catégorie mise à jour
3 Documentation de deuxième catégorie inchangée
4 Mise à jour payante de la documentation de première catégorie
5 Mise à jour gratuite de la documentation de première catégorie
6 Nouveautés payantes non vitales
7 Nouveautés gratuites et vitales

Taille : 4 page(s) environ

Domaine : 816

VERSION : rev 2
DATE : 20.11.85

Numeric Magnitude Comparisons on the 65816

To:	Gumby Team
From:	Jim Jatczynski
Date:	First draft: 11/13/85; Revision 1: 11/14/85; Revision 2: 11/20/85

SHOULD I READ THIS MEMO?

Consider the following code for testing p<q where p and q are 16-bit two's complement integers:

```
lda   p
cmp   q
bmi   dest
```

If you think this works, you should probably read this memo. It describes correct methods of performing signed and unsigned arithmetic comparisons on the 65816. If you have read previous versions of this memo, throw them away and save this one—it has additional routines.

NOTATION

Let n, z, v, and c be the negative, zero, overflow, and carry bit values and ~n, ~z, ~v, and ~c be their complements. Adjacency indicates logical and and "+" indicates logical or.

DID YOU KNOW SBC AND CMP ARE NOT THE SAME?

Subtraction affects the n, z, v, and c status bits; comparison affects only the n, z, and c bits. As you will see, knowing the value of v is crucial in certain comparisons. Such comparisons must use SBC rather than CMP to compare magnitudes of the operands.

SIGNED TWO'S COMPLEMENT COMPARISON

We want to compare two 16-bit two's complement integers, p and q, and branch to location dest if one of the conditions <, ≤, =, ≠, ≥, or > is met. We can set the condition codes by executing

```
lda   p
sec
sbc   q
{ conditional branch code }
```

The following table shows the appropriate branch conditions in terms of the n, z, v and c statusbits:

p < q	n~v + ~nv
p ≤ q	z + n~v + ~nv
p = q	z
p ≠ q	~z
p ≥ q	nv + ~n~v
p > q	nv~z + ~n~v~z

It is immediately clear from this table why the code shown in the first section doesn't work. First, CMP doesn't even affect the v flag. But, even if it did, the branch instruction BMI only looks at the n flag which will only be correct if there is no overflow during the subtraction.

So, correct code for "**if** p < q **then go to** dest **else** continue" might look like this:

```
          lda    p             ;load p
          sec                  ;preset carry for proper subtraction
          sbc    q             ;compare and set condition codes
          bvs    lab1          ;check v
          bmi    dest          ;here ~v, check n
          bra    continue      ;here ~v~n
lab1      bpl    dest          ;here v, check n
continue                       ;here the branch condition is not met
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
dest                           ;here the branch condition is met, i.e.
                               ;n~v + ~nv is true
```

Rich Williams has pointed out a neat alternative to this code that saves 1 byte in native mode and 2 bytes in 8-bit mode:

```
          lda    p             ;branch if p<q
          sec                  ;preset carry for proper subtraction
          sbc    q             ;compare and set condition codes
          bvs    lab1          ;check v, branch if set
          eor    #$8000        ;here ~v, flip sign of result
lab1      bpl    dest          ;check n and v have different values
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
dest
```

Code similar to the first example can be written for all the other branch conditions. For example, the code for ≥ is as follows:

```
          lda    p             ;branch if p≥q
          sec
          sbc    q
          bvs    lab1
          bpl    dest
          bra    continue
lab1      bmi    dest
continue
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
dest                           ;here the branch condition is met, i.e.
                               ;nv + ~n~v
```

Rich's method can also be applied to this branch condition as follows:

```
          lda    p             ;branch if p≥q
          sec
          sbc    q
          bvs    lab1
          eor    #$8000
lab1      bmi    dest
. . . . . . . . . . . . .
dest
```

The code for = and ≠ is simple because each involves only the z bit.

```
          lda    p             ;branch if p=q
          cmp    q             ;can use CMP because only z is needed
          beq    dest

          lda    p             ;branch if p≠q
          cmp    q
          bne    dest
```

The code for > and ≤ is more complex because it involves the n, z, and v bits. Rich's method can be written as follows:

```
          lda    p            ;branch if p>q
          sec
          sbc    q
          beq    continue     ;if equal, don't branch to dest
          bvs    lab1
          eor    #$8000
lab1      bmi    dest
continue
. . . . . . . . . . . . .
dest

          lda    p            ;branch if p≤q
          sec
          sbc    q
          beq    dest         ;if equal, branch to dest now
          bvs    lab1
          eor    #$8000       ;flip sign of result
lab1      bpl    dest
. . . . . . . . . . . . . . . . . . . . . . .
dest
```

The code above is actually most applicable if operand p is already in the accumulator. When this isn't the case, it may be more efficient to reverse the operands and use the opposite branch condition to get more faster code. For example, the code for < is generally faster than the code for >, and the code for ≥ is generally faster than the code for ≤.

UNSIGNED COMPARISON

Unsigned comparisons can be done using the sequence "`LDA p, CMP q`" because the v bit is not involved in the comparison conditions which are as follows:

p < q	~c
p ≤ q	[see below]
p = q	z
p ≠ q	~z
p ≥ q	c
p > q	[see below]

Thus, the code for < and ≥ is very simple:

```
      lda    p
      cmp    q
      bcc    dest      ;branch to dest if p<q

      lda    p
      cmp    q
      bcs    dest      ;branch to dest if p≥q
```

Many assemblers have the aliases `blt` and `bge` for `bcc` and `bcs`, respectively, to make code reading easier.

The easiest way to test the conditions ≤ and > is to reverse the operands and use the code above. For example, to test p≤q, use the following:

```
        lda   q
        cmp   p
        bcs   dest      ;branch to dest if p≤q
```

MULTIPLE PRECISION COMPARISONS

The code for multiple precision comparisons is more complicated, so we don't show any of it here. A good reference for multiple precision comparison routines is Leventhal's and Saville's *6502 Assembly Language Subroutines*, OSBORNE/McGraw-Hill.