Apple Assembly Line - How to Add and Subtract One#

General Information

Author: Bob Sander-Cederlof
Assembler: generic
Published: October 1980, Issue 1
Download: http://www.txbobsc.com/aal/

I suppose there are as many ways to do it as there are programmers. Some are short and fast, some long and slow, some neat, some sloppy.

Adding one to a number is called "incrementing", and subtracting one is called "decrementing". The 6502 has two instructions for these two functions: INC and DEC. (For the moment I will overlook the four instructions for doing the same to the X and Y registers: INX, INY, DEX, and DEY.) It is easy to see how to use them on single-byte values; with a little more trouble we can also use them for values of two or more bytes.

Single-Byte Values:#

Here are five different ways to increment a single byte:

Methods 1 and 2: Add 1

```
CLC                 SEC
LDA VALUE           LDA VALUE
ADC #1              ADC #0
STA VALUE           STA VALUE
```

Method 3 and 4: Subtract (-1)

```
SEC                 CLC
LDA VALUE           LDA VALUE
SBC #$FF             SBC #$FE
STA VALUE           STA VALUE
```

Method 5: Use the INC instruction

```
INC VALUE
```

Here are five similar ways to decrement a value:

Method 1 and 2: Subtract 1

```
SEC                 CLC
LDA VALUE           LDA VALUE
SBC #1              SBC #0
STA VALUE           STA VALUE
```

Method 3 and 4: Add (-1)

```
CLC                 SEC
LDA VALUE           LDA VALUE
ADC #$FF             ADC #$FE
STA VALUE           STA VALUE
```

Method 5: Use the DEC instruction

```
      DEC VALUE
```

There are times when any of the above may be justified, depending on the state of the A-register and the Carry Status bit.

Multi-Byte Values:#

Incrementing a two-byte value is a very common practice in 6502 programs. Here are two methods:

Method 1: Add 1

```
      CLC
      LDA VALL  LOW BYTE
      ADC #1
      STA VALL
      LDA VALH  HIGH BYTE
      ADC #0
      STA VALH
```

Method 2: Use the INC instruction

```
      INC VALL  INCREMENT LOW BYTE
      BNE .1      IF NOT ZERO, THEN NO CARRY
      INC VALH  INCREMENT HIGH BYTE
.1  .....
```

Of course, there are many variations on these methods. It is easy to see how to extend these two methods to more than two bytes. Here is a three-byte version of Method 2:

```
      INC VALL  INCREMENT LOW BYTE
      BNE .1      UNLESS ZERO, NO CARRY
      INC VALM  INCREMENT MIDDLE BYTE
      BNE .1      UNLESS ZERO, NO FURTHER CARRY
      INC VALH  INCREMENT HIGH BYTE
.1  ....
```

Believe it or not, there is one disadvantage to using Method 2, in some circumstances. Sometimes code is required to have a constant running time; then, Method 1 is the one to use. But most of the time, Method 2 is the best.

How about subtracting one? Here are two ways to do it to a two-byte value:

Method 1: Subtract 1

```
      SEC
      LDA VALL
      SBC #1
      STA VALL
      LDA VALH
      SBC #0
      STA VALH
```

Method 2: Use the DEC instruction

```
      LDA VALL  SEE IF NEED TO BORROW
```

```
      BNE .1      NO
      DEC VALH  YES
.1    DEC VALL
```

Which one do you like better? It is still a matter of taste, unless the amount of memory used or time consumed is very important. There are also different side effects, such as the final state of the carry status. INC and DEC do not change the carry status, while of course ADC and SBC do. You may wish to preserve carry through the process, making the INC/DEC code preferable. Or, you may wish to know the resulting carry status after incrementing or decrementing for some reasong; then you should use the ADC/SBC code.

Back to subtracting one...how about doing it to a three-byte value? We just add three more lines:

```
      LDA VALL  SEE IF NEED TO BORROW
      BNE .2      NO
      LDA VALM  SEE IF NEED TO BORROW AGAIN
      BNE .1      NO
      DEC VALH  BORROW FROM HIGH BYTE
.1    DEC VALM  BORROW FROM MIDDLE BYTE
.2    DEC VALL
```

Easier than you though, right? You would not believe the many strange ways I have seen this operation coded in commercial software (even some released by Apple themselves!). Yet it seems to me that this method is the same way we would do it with pencil and paper in decimal arithmetic. Think how you would do this:

```
123040
    -1
------
xxxxxx
```

If you think of each digit as though it were a byte...isn't the algorithm the same?

Now it is time for all of us to go back over the programs we wrote during the past three years for the Apple, and replace a lot of old code!

Bob Sander-Cederlof

Bob Sander-Cederlof | 19.11.2007 at 03:57 PM

Thank you for republishing my article. The Apple Assembly Line newsletter was published from monthly October 1980 through May 1988. All the issues are available online at http://www.txbobsc.com/aal/