Hitachi America, Ltd.

May, 1991

Application Engineering
TechNotes

Tom Hampton

H8/300 CPU SUBX Instruction

The H8/300 CPU provides an instruction for subtracting two bytes from each other along with the value of the Carry flag. This instruction is useful when performing subtraction operations that are greater than 16-bits (an instruction is already available that can do either 8-bit or 16-bit subtractions with no problems). Lets take the example of a 32-bit subtraction as follows:

```
  H'40000000
- H'3F8F2356
  H'0070DCAA          (result)
```

If we look are each operation individually, the result is easily explained.

1. In subtracting the low order bytes from each other (56 from 00), we get a result of AA with a borrow from the next higher byte.
2. In subtracting the next higher order bytes from each other (23 from FF because of the borrow), we get a result of DC with the borrow continuing to the next higher byte.
3. In subtracting the next higher order bytes from each other (8F from FF because of the borrow), we get a result of 70 with the borrow continuing to the next higher byte.
4. In subtracting the highest order bytes from each other (3F from 3F because of the borrow), we get a result of 00 with no borrow.

If no borrow operations were never to occur, then we could code this very simply with two word subtract operations. But since this is not the case, we must code the sequence so as to keep track of the borrow operations. If we code this in the same sequence as the operation described above, it might look something like this:

```
mov.w     #h'4000,r1
mov.w     #0,r2
mov.w     #h'3f8f,r3
mov.w     #h'2356,r4
sub.b     r2l,r4l             ;00-56
subx      r2h,r4h             ;00-23-borrow
subx      r1l,r3l             ;00-8f-borrow
subx      r1h,r3l             ;40-3f-borrow
```

(We could also replace the first two subtraction instructions with a subtract word operation to reduce code size and execution time but this method makes it easier to read for now.)

```
sub.w     r2,r4               ;0000-2356
subx      r1l,r3l             ;00-8f-borrow
subx      r1h,r3l             ;40-3f-borrow
```

The trick in using the SUBX instruction is to pay attention to the flag operations. During execution of a normal subtraction operation, the Zero flag is used to determine if the result of the operation is zero or not. However, the execution of the SUBX instruction is a little bit different. If the result of the operation is zero, then the Zero flag remains unchanged from the previous instruction. If the result is non-zero, then the Zero flag is cleared to correctly indicate a non-zero result. While this sounds a lot like what it is supposed to be, look at a scenario where the previous instruction would clear the zero flag (this may be something as simple as a MOV instruction). If the SUBX instruction were to follow this operation and the result were zero, then the Zero flag would remain at "0," clearly not indicating the result of the operation.

Because of this, it is extremely important that the SUBX instruction be used **IMMEDIATELY** following other SUB or SUBX instructions. This sequence allows the H8/300 CPU to properly keep track of borrows, and maintain the Zero flag in the correct state. To illustrate this problem, lets assume that our variables are stored in memory rather than registers. In this example, we have to move the data into our registers in order to perform the operation.

```
   var1     .equ     H'40000000
   var2     .equ     H'3f8f2356
            mov.b    @(var2+3),r1h       ;get 56
            mov.b    @(var1+3),r1l       ;get 00
1.          sub.b    r1h,r1l             ;00-56, Zero=0, Carry=1
            mov.b    r1l,@(var1+3)       ;store 1st result (AA), Zero=0, Carry=1
            mov.b    @(var2+2),r1h       ;get 23, Zero=0, Carry=1
            mov.b    @(var1+2),r1l       ;get 00, Zero=1, Carry=1
2.          subx     r1h,r1l             ;00-23-borrow, Zero=1, Carry=1
            mov.b    r1l,@(var1+2)       ;store 2nd result (DC), Zero=0, Carry=1
            mov.b    @(var2+1),r1h       ;get 8F, Zero=0, Carry=1
            mov.b    @(var1+1),r1l       ;get 00, Zero=1, Carry=1
3.          subx     r1h,r1l             ;00-8F-borrow, Zero=1, Carry=1
            mov.b    r1l,@(var1+1)       ;store 2nd result (70), Zero=0, Carry=1
            mov.b    @(var2),r1h         ;get 3F, Zero=0, Carry=1
            mov.b    @(var1),r1l         ;get 40, Zero=0, Carry=1
4.          subx     r1h,r1l             ;40-3F-borrow, Zero=0, Carry=0
            mov.b    r1l,@(var1)         ;store 2nd result (00), Zero=1, Carry=0
```

Since the result of the SUB and SUBX operations are not 00 (until number 4), the flags behave as we wish them to. During the 4th subtraction operation, the Zero flag remains clear even though the result of the subtraction operation was zero. While this is the correct flag value for the entire operation, it would not be correct if we test the flag after the MOV instruction.

Lets change our variables so that the result of the subtraction operation should be zero (H'40000000 - H'40000000) and again follow the code sequence. Since each of the substraction operations (SUBX) would result in a zero value, the contents of the Zero flag would remain as it was prior to the execution of the instruction. In the 4th subtraction operation, we need only look at the values transfered by the MOV instructions. Since both values are non-zero in nature, then the contents of the Zero flag would be cleared as a result of that instruction. This allows the final value of the Zero flag (before the MOV instruction) to be 0, and this would be incorrect for the entire operation. Of course it would be correct after the MOV operation, but our previous example showed it to be opposite of this example.

```
                mov.b     r1l,@(var1+1)        ;store 2nd result (70), Zero=0, Carry=1
                mov.b     @(var2),r1h          ;get 40, Zero=0, Carry=1
                mov.b     @(var1),r1l          ;get 40, Zero=0, Carry=1
        4.      subx      r1h,r1l              ;40-3F-borrow, Zero=0, Carry=0
                mov.b     r1l,@(var1)          ;store 2nd result (00), Zero=1, Carry=0
```

In the final analysis, to make things much simpler for the user, it is recommended that the SUBX instructions always follow other subtraction instructions **IMMEDIATELY**. The resulting Zero flag status would also the user to determine the status of his complete result.