```
module ripple_carry_adder_subtractor(S, C, V, A, B, 
Op);
   output [3:0] S;   // The 4-bit sum/difference.
   output   C;   // The 1-bit carry/borrow status.
   output   V;   // The 1-bit overflow status.
   input [3:0]  A;   // The 4-bit augend/minuend.
   input [3:0]  B;   // The 4-bit addend/subtrahend.
   input    Op;  // The operation: 0 => Add, 
1=>Subtract.

   wire C0; // The carry out bit of fa0, the carry 
in bit of fa1.
   wire C1; // The carry out bit of fa1, the carry 
in bit of fa2.
   wire C2; // The carry out bit of fa2, the carry 
in bit of fa3.
   wire C3; // The carry out bit of fa2, used to 
generate final carry/borrow.

   wire B0; // The xor'd result of B[0] and Op
   wire B1; // The xor'd result of B[1] and Op
   wire B2; // The xor'd result of B[2] and Op
   wire B3; // The xor'd result of B[3] and Op



   // Looking at the truth table for xor we see that 
   // B xor 0 = B, and
   // B xor 1 = not(B).
   // So, if Op==1 means we are subtracting, then
   // adding A and B xor Op alog with setting the 
first
   // carry bit to Op, will give us a result of
   // A+B when Op==0, and A+not(B)+1 when Op==1.
   // Note that not(B)+1 is the 2's complement of B, 
so
```

```
   // this gives us subtraction.
   xor(B0, B[0], Op);
   xor(B1, B[1], Op);
   xor(B2, B[2], Op);
   xor(B3, B[3], Op);
   xor(C, C3, Op);     // Carry = C3 for addition,
Carry = not(C3) for subtraction.
   xor(V, C3, C2);     // If the two most significant
carry output bits differ, then we have an overflow.

   full_adder fa0(S[0], C0, A[0], B0, Op);    //
Least significant bit.
   full_adder fa1(S[1], C1, A[1], B1, C0);
   full_adder fa2(S[2], C2, A[2], B2, C1);
   full_adder fa3(S[3], C3, A[3], B3, C2);    // Most
significant bit.
endmodule // ripple_carry_adder_subtractor
```