

SIGN EXTENSION

Sign Extension

- Rule #1: 2's complement input and output operands must be the same word-width because of implied zeros

$$\begin{array}{r} 001011 \quad -5 \\ 010010 \quad +18 \\ \hline 011101 \quad +29! \end{array}$$

Sign Extension

- Rule #2: Despite a fundamental change to the number's definition, the value of a 2's complement number will never change due to any amount of sign extension—regardless of whether the value is positive or negative

$$\begin{array}{rcl} & 1 & 0 & 1 & 1 & -5 = -8 + 2 + 1 \\ \mathbf{1} & 1 & 0 & 1 & 1 & -5 = -16 + 8 + 2 + 1 \\ \mathbf{1} & \mathbf{1} & 1 & 0 & 1 & 1 & -5 = -32 + 16 + 8 + 2 + 1 \end{array}$$

Sign Extension

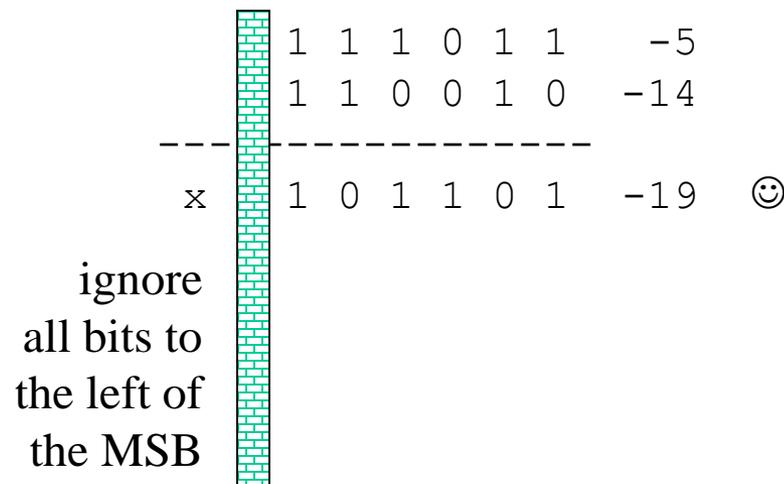
- Procedure:
 - 1) Calculate the necessary minimum width of the sum so that it contains all input possibilities
 - It's up to you to make sure the output range is sufficient
 - 2) Extend the inputs' sign bits to the width of the answer
 - 3) Add as usual
 - 4) Ignore bits that ripple to the left of the answer's MSB

1	1	1	1	1	0	1	1	-5	[-8,+7]	4-bit
0	0	1	0	0	1	0	0	+18	[-32,+31]	6-bit
x	0	0	0	1	1	0	1	+13	☺	[-40,+38]
										Choose 7-bit [-64,+63]



Sign Extension

- Ignore carry bits
 - Do not spend any hardware calculating any bits to the left of the answer's MSB



Sign Extension In Verilog

- Adding signed variables in verilog requires one of two methods:
 - Using *signed* variables: always works in verilog simulators but in rare cases these variables do not work as expected with some CAD tools
 - Declare all variables normally as *regs* and *wires* and perform sign extension manually. This method can be tedious but will *always always* work as expected.

Sign Extension In Verilog

- These cases use no sign extension at all
- Two cases work correctly, one does not

```
reg [3:0] m, n;
reg [4:0] sum;

initial begin
    m = 4'b0000;
    n = 4'b0000;
    sum = m + n;
    #10;
    $write("m = %b, n = %b, sum = %b\n", m, n, sum);

    m = 4'b1111;
    n = 4'b1111;
    sum = m + n;
    #10;
    $write("m = %b, n = %b, sum = %b\n", m, n, sum);

    m = 4'b1111;
    n = 4'b0000;
    sum = m + n;
    #10;
    $write("m = %b, n = %b, sum = %b # ERROR: -1 + 0 = +15\n", m, n, sum);
end
```

```
m = 0000, n = 0000, sum = 00000
m = 1111, n = 1111, sum = 11110 # Lucky!
m = 1111, n = 0000, sum = 01111 # ERROR: -1 + 0 = +15
```

Sign Extension In Verilog

- Writing the sign extension manually is the most robust method
- Both inputs are sign extended to the same width as the sum—five bits in this case

```
reg [3:0] m, n;  
reg [4:0] sum;  
  
initial begin  
    m = 4'b1111;  
    n = 4'b0000;  
    sum = {m[3],m} + {n[3],n};  
    #10;  
    $write("m = %b, n = %b, sum = %b # OK!\n", m, n, sum);  
end
```

```
m = 1111, n = 0000, sum = 11111 # OK!
```