```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/HalfAdder.hdl

/**
 * Computes the sum of two bits.
 */

CHIP HalfAdder {
    IN a, b;    // 1-bit inputs
    OUT sum,    // Right bit of a + b
        carry;  // Left bit of a + b

    PARTS:
    Xor( a = a, b = b, out = sum);
    And( a = a, b = b, out = carry);
}
```

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/FullAdder.hdl

/**
 * Computes the sum of three bits.
 */

CHIP FullAdder {
    IN a, b, c;  // 1-bit inputs
    OUT sum,     // Right bit of a + b + c
        carry;   // Left bit of a + b + c

    PARTS:
    HalfAdder( a = a, b = b, sum = ab, carry = w1);
    HalfAdder( a = ab, b = c, sum = sum, carry = w2);
    Or( a = w1, b = w2, out = carry);
}
```

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/Adder16.hdl

/**
 * Adds two 16-bit values.
 * The most significant carry bit is ignored.
 */

CHIP Add16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    HalfAdder( a = a[0], b = b[0], sum = out[0], carry = w1);
    FullAdder( a = a[1], b = b[1], c = w1, sum = out[1], carry = w2);
    FullAdder( a = a[2], b = b[2], c = w2, sum = out[2], carry = w3);
    FullAdder( a = a[3], b = b[3], c = w3, sum = out[3], carry = w4);
    FullAdder( a = a[4], b = b[4], c = w4, sum = out[4], carry = w5);
    FullAdder( a = a[5], b = b[5], c = w5, sum = out[5], carry = w6);
    FullAdder( a = a[6], b = b[6], c = w6, sum = out[6], carry = w7);
    FullAdder( a = a[7], b = b[7], c = w7, sum = out[7], carry = w8);
    FullAdder( a = a[8], b = b[8], c = w8, sum = out[8], carry = w9);
    FullAdder( a = a[9], b = b[9], c = w9, sum = out[9], carry = w10);
    FullAdder( a = a[10], b = b[10], c = w10, sum = out[10], carry = w11);
    FullAdder( a = a[11], b = b[11], c = w11, sum = out[11], carry = w12);
    FullAdder( a = a[12], b = b[12], c = w12, sum = out[12], carry = w13);
    FullAdder( a = a[13], b = b[13], c = w13, sum = out[13], carry = w14);
    FullAdder( a = a[14], b = b[14], c = w14, sum = out[14], carry = w15);
    FullAdder( a = a[15], b = b[15], c = w15, sum = out[15]);
}
```

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/Inc16.hdl

/**
 * 16-bit incrementer:
 * out = in + 1 (arithmetic addition)
 */

CHIP Inc16 {
    IN in[16];
    OUT out[16];

    PARTS:
    Add16( a = in, b[0] = true, out = out);
}
```

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/02/ALU.hdl

/**
 * The ALU (Arithmetic Logic Unit).
 * Computes one of the following functions:
 * x+y, x-y, y-x, 0, 1, -1, x, y, -x, -y, !x, !y,
 * x+1, y+1, x-1, y-1, x&y, x|y on two 16-bit inputs,
 * according to 6 input bits denoted zx,nx,zy,ny,f,no.
 * In addition, the ALU computes two 1-bit outputs:
 * if the ALU output == 0, zr is set to 1; otherwise zr is set to 0;
 * if the ALU output < 0, ng is set to 1; otherwise ng is set to 0.
 */

// Implementation: the ALU logic manipulates the x and y inputs
// and operates on the resulting values, as follows:
// if (zx == 1) set x = 0        // 16-bit constant
// if (nx == 1) set x = !x       // bitwise not
// if (zy == 1) set y = 0        // 16-bit constant
// if (ny == 1) set y = !y       // bitwise not
// if (f == 1)  set out = x + y  // integer 2's complement addition
// if (f == 0)  set out = x & y  // bitwise and
// if (no == 1) set out = !out   // bitwise not
// if (out == 0) set zr = 1
// if (out < 0) set ng = 1

CHIP ALU {
    IN
        x[16], y[16],  // 16-bit inputs
        zx, // zero the x input?
        nx, // negate the x input?
        zy, // zero the y input?
        ny, // negate the y input?
        f,  // compute out = x + y (if 1) or x & y (if 0)
        no; // negate the out output?

    OUT
        out[16], // 16-bit output
        zr, // 1 if (out == 0), 0 otherwise
        ng; // 1 if (out < 0),  0 otherwise

    PARTS:
    Mux16( a = x, b = false, sel = zx, out = x1);
    Not16( in = x1, out = notx1);
    Mux16( a = x1, b = notx1, sel = nx, out = x2);

    Mux16( a = y, b = false, sel = zy, out = y1);
    Not16( in = y1, out = noty1);
    Mux16( a = y1, b = noty1, sel = ny, out = y2);
    And16( a = x2, b = y2, out = xy1);
    Add16( a = x2, b = y2, out = xy2);
    Mux16( a = xy1, b = xy2, sel = f, out = out1);

    Not16( in = out1, out = notout1);
    Mux16( a = out1, b = notout1, sel = no, out[0..7] = or1, out[8..15] = or2, out[15]
= ng, out = out);

    Or8Way( in = or1, out = orout1);
    Or8Way( in = or2, out = orout2);
    Or( a = orout1, b = orout2, out = notzr);
    Not( in = notzr, out = zr);
}
```