```
CHIP Not {
    IN in;
    OUT out;

    PARTS:
    Nand (a=in, b=in, out=out);
}

CHIP And {
    IN a, b;
    OUT out;

    PARTS:
    Nand (a=a, b=b, out=x);
    Not  (in=x, out=out);
}

CHIP Or {
    IN a, b;
    OUT out;

    PARTS:
    Not  (in=a, out=nota);
    Not  (in=b, out=notb);
    Nand (a=nota, b=notb, out=out);
}

CHIP Xor {
    IN a, b;
    OUT out;

    PARTS:
    // The canonical representation as presented in the book
    Not (in=a, out=nota);
    Not (in=b, out=notb);
    And (a=a, b=notb, out=term1);
    And (a=nota, b=b, out=term2);
    Or  (a=term1, b=term2, out=out);
}

CHIP Mux {
    IN a, b, sel;
    OUT out;

    PARTS:
    Not (in=sel, out=notSel);
    And (a=notSel, b=a, out=and1);
    And (a=sel, b=b, out=and2);
    Or  (a=and1, b=and2, out=out);
}

CHIP DMux {
    IN in, sel;
    OUT a, b;

    PARTS:
    Not (in=sel, out=notSel);
    And (a=in, b=notSel, out=a);
    And (a=in, b=sel, out=b);
}

CHIP Not16 {
    IN in[16];
    OUT out[16];

    PARTS:
    Not (in=in[0],  out=out[0]);
    Not (in=in[1],  out=out[1]);
    Not (in=in[2],  out=out[2]);
    Not (in=in[3],  out=out[3]);
    Not (in=in[4],  out=out[4]);
    Not (in=in[5],  out=out[5]);
    Not (in=in[6],  out=out[6]);
    Not (in=in[7],  out=out[7]);
    Not (in=in[8],  out=out[8]);
    Not (in=in[9],  out=out[9]);
    Not (in=in[10], out=out[10]);
    Not (in=in[11], out=out[11]);
    Not (in=in[12], out=out[12]);
    Not (in=in[13], out=out[13]);
    Not (in=in[14], out=out[14]);
    Not (in=in[15], out=out[15]);
}

CHIP And16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    And (a=a[0],  b=b[0],  out=out[0]);
    And (a=a[1],  b=b[1],  out=out[1]);
    And (a=a[2],  b=b[2],  out=out[2]);
    And (a=a[3],  b=b[3],  out=out[3]);
    And (a=a[4],  b=b[4],  out=out[4]);
    And (a=a[5],  b=b[5],  out=out[5]);
    And (a=a[6],  b=b[6],  out=out[6]);
    And (a=a[7],  b=b[7],  out=out[7]);
    And (a=a[8],  b=b[8],  out=out[8]);
    And (a=a[9],  b=b[9],  out=out[9]);
    And (a=a[10], b=b[10], out=out[10]);
    And (a=a[11], b=b[11], out=out[11]);
    And (a=a[12], b=b[12], out=out[12]);
    And (a=a[13], b=b[13], out=out[13]);
    And (a=a[14], b=b[14], out=out[14]);
    And (a=a[15], b=b[15], out=out[15]);
}

CHIP Or16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    Or (a=a[0],  b=b[0],  out=out[0]);
    Or (a=a[1],  b=b[1],  out=out[1]);
    Or (a=a[2],  b=b[2],  out=out[2]);
    Or (a=a[3],  b=b[3],  out=out[3]);
    Or (a=a[4],  b=b[4],  out=out[4]);
    Or (a=a[5],  b=b[5],  out=out[5]);
    Or (a=a[6],  b=b[6],  out=out[6]);
    Or (a=a[7],  b=b[7],  out=out[7]);
    Or (a=a[8],  b=b[8],  out=out[8]);
    Or (a=a[9],  b=b[9],  out=out[9]);
    Or (a=a[10], b=b[10], out=out[10]);
    Or (a=a[11], b=b[11], out=out[11]);
    Or (a=a[12], b=b[12], out=out[12]);
    Or (a=a[13], b=b[13], out=out[13]);
    Or (a=a[14], b=b[14], out=out[14]);
    Or (a=a[15], b=b[15], out=out[15]);
}

CHIP Mux16 {
    IN a[16], b[16], sel;
    OUT out[16];

    PARTS:
    Mux (a=a[0],  b=b[0],  sel=sel, out=out[0]);
    Mux (a=a[1],  b=b[1],  sel=sel, out=out[1]);
    Mux (a=a[2],  b=b[2],  sel=sel, out=out[2]);
    Mux (a=a[3],  b=b[3],  sel=sel, out=out[3]);
    Mux (a=a[4],  b=b[4],  sel=sel, out=out[4]);
    Mux (a=a[5],  b=b[5],  sel=sel, out=out[5]);
    Mux (a=a[6],  b=b[6],  sel=sel, out=out[6]);
    Mux (a=a[7],  b=b[7],  sel=sel, out=out[7]);
    Mux (a=a[8],  b=b[8],  sel=sel, out=out[8]);
    Mux (a=a[9],  b=b[9],  sel=sel, out=out[9]);
    Mux (a=a[10], b=b[10], sel=sel, out=out[10]);
    Mux (a=a[11], b=b[11], sel=sel, out=out[11]);
    Mux (a=a[12], b=b[12], sel=sel, out=out[12]);
    Mux (a=a[13], b=b[13], sel=sel, out=out[13]);
    Mux (a=a[14], b=b[14], sel=sel, out=out[14]);
    Mux (a=a[15], b=b[15], sel=sel, out=out[15]);
}

CHIP Or8Way {
    IN in[8];
    OUT out;

    PARTS:
    // Binary tree of Or gates.
    Or (a=in[0], b=in[1], out=or01);
    Or (a=in[2], b=in[3], out=or23);
    Or (a=in[4], b=in[5], out=or45);
    Or (a=in[6], b=in[7], out=or67);
    Or (a=or01, b=or23, out=or0123);
    Or (a=or45, b=or67, out=or4567);
    Or (a=or0123, b=or4567, out=out);
}

CHIP Mux4Way16 {
    IN a[16], b[16], c[16], d[16], sel[2];
    OUT out[16];

    PARTS:
    Mux16 (a=a,  b=b,  sel=sel[0], out=ab);
    Mux16 (a=c,  b=d,  sel=sel[0], out=cd);
    Mux16 (a=ab, b=cd, sel=sel[1], out=out);
}

CHIP Mux8Way16 {
    IN a[16], b[16], c[16], d[16],
       e[16], f[16], g[16], h[16],
       sel[3];
    OUT out[16];

    PARTS:
    // Binary tree of 2-way multiplexors
    Mux16 (a=a,    b=b,    sel=sel[0], out=ab);
    Mux16 (a=c,    b=d,    sel=sel[0], out=cd);
    Mux16 (a=e,    b=f,    sel=sel[0], out=ef);
    Mux16 (a=g,    b=h,    sel=sel[0], out=gh);
    Mux16 (a=ab,   b=cd,   sel=sel[1], out=abcd);
    Mux16 (a=ef,   b=gh,   sel=sel[1], out=efgh);
    Mux16 (a=abcd, b=efgh, sel=sel[2], out=out);
}

CHIP DMux4Way {
    IN in, sel[2];
    OUT a, b, c, d;

    PARTS:
    DMux (in=in, sel=sel[1], a=ab, b=cd);
    DMux (in=ab, sel=sel[0], a=a,  b=b);
    DMux (in=cd, sel=sel[0], a=c,  b=d);
}


CHIP DMux8Way {
    IN in, sel[3];
    OUT a, b, c, d, e, f, g, h;

    PARTS:
    // Binary tree of demultiplexors.
    DMux (sel=sel[2], in=in,   a=abcd, b=efgh);
    DMux (sel=sel[1], in=abcd, a=ab,   b=cd);
    DMux (sel=sel[1], in=efgh, a=ef,   b=gh);
    DMux (sel=sel[0], in=ab,   a=a,    b=b);
    DMux (sel=sel[0], in=cd,   a=c,    b=d);
    DMux (sel=sel[0], in=ef,   a=e,    b=f);
    DMux (sel=sel[0], in=gh,   a=g,    b=h);
}
```