```
/** Checks if two 3-bit input buses are equal */
CHIP EQ3 {
IN a[3], b[3];
OUT out; // True iff a=b PARTS:
Xor(a=a[0], b=b[0], out=c0);
Xor(a=a[1], b=b[1], out=c1);
Xor(a=a[2], b=b[2], out=c2);
Or(a=c0, b=c1, out=c01);
Or(a=c01, b=c2, out=neq); Not(in=neq, 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/01/And.hdl

/**
 * And gate:
 * out = 1 if (a == 1 and b == 1)
 *       0 otherwise
 */

CHIP And {
    IN a, b;
    OUT out;

    PARTS:
    Nand(a=a, b=b, out=c);
    Not(in=c,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/01/And16.hdl

/**
 * 16-bit bitwise And:
 * for i = 0..15: out[i] = (a[i] and b[i])
 */

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]);
}// 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/01/DMux.hdl

/**
 * Demultiplexor:
 * {a, b} = {in, 0} if sel == 0
 *          {0, in} if sel == 1
 */

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

    PARTS:
    Not(in=sel,out=sel0);
    And(a=sel, b=in, out=b);
    And(a=sel0, b=in, out=a);
}
// 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/01/DMux4Way.hdl

/**
 * 4-way demultiplexor:
 * {a, b, c, d} = {in, 0, 0, 0} if sel == 00
 *                {0, in, 0, 0} if sel == 01
 *                {0, 0, in, 0} if sel == 10
 *                {0, 0, 0, in} if sel == 11
 */

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

    PARTS:
    DMux(in=in, sel=sel[1], a=g1, b=g2);
    DMux(in=g1, sel=sel[0], a=a , b=b);
    DMux(in=g2, sel=sel[0], a=c , b=d);
```

```
}// 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/01/DMux8Way.hdl

/**
 * 8-way demultiplexor:
 * {a, b, c, d, e, f, g, h} = {in, 0, 0, 0, 0, 0, 0, 0} if sel == 000
 *                            {0, in, 0, 0, 0, 0, 0, 0} if sel == 001
 *                            etc.
 *                            {0, 0, 0, 0, 0, 0, 0, in} if sel == 111
 */

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

    PARTS:
    DMux(in=in, sel=sel[2], a=g1, b=g2);
    DMux4Way(in=g1, sel=sel[0..1], a=a , b=b, c=c, d=d);
    DMux4Way(in=g2, sel=sel[0..1], a=e , b=f, c=g, d=h);
}// 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/01/Mux.hdl

/**
 * Multiplexor:
 * out = a if sel == 0
 *       b otherwise
 */

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

    PARTS:
    Not(in=sel,out=sel0);
    And(a=a, b=sel0, out=c1);
    And(a=b, b=sel, out=c2);
    Or(a=c1, b=c2, out=out);
    // Put your code here:
}// 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/01/Mux16.hdl

/**
 * 16-bit multiplexor:
```

```
 * for i = 0..15 out[i] = a[i] if sel == 0
 *                        b[i] if sel == 1
 */

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]);
}
// 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/01/Mux4Way16.hdl

/**
 * 4-way 16-bit multiplexor:
 * out = a if sel == 00
 *       b if sel == 01
 *       c if sel == 10
 *       d if sel == 11
 */

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=x);
    Mux16(a=c,b=d,sel=sel[0],out=y);
    Mux16(a=x,b=y,sel=sel[1],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/01/Mux8Way16.hdl

/**
 * 8-way 16-bit multiplexor:
 * out = a if sel == 000
 *       b if sel == 001
 *       etc.
 *       h if sel == 111
 */

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:
    Mux4Way16(a=a,b=b,c=c,d=d,sel=sel[0..1],out=x);
    Mux4Way16(a=e,b=f,c=g,d=h,sel=sel[0..1],out=y);
    Mux16(a=x,b=y,sel=sel[2],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/01/Not.hdl

/**
 * Not gate:
 * out = not in
 */

CHIP Not {
    IN in;
    OUT out;

    PARTS:
    Nand(a=in,b=in,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/01/Not16.hdl

/**
 * 16-bit Not:
 * for i=0..15: out[i] = not in[i]
 */

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]);

}// 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/01/Or.hdl

 /**
 * Or gate:
 * out = 1 if (a == 1 or b == 1)
 *       0 otherwise
 */

CHIP Or {
    IN a, b;
    OUT out;

    PARTS:
    Nand(a=a, b=a, out=c1);
    Nand(a=b, b=b, out=c2);
    Nand(a=c1, b=c2, 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/01/Or16.hdl

/**
 * 16-bit bitwise Or:
 * for i = 0..15 out[i] = (a[i] or b[i])
 */

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]);
}// 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/01/Or8Way.hdl

/**
 * 8-way Or:
 * out = (in[0] or in[1] or ... or in[7])
 */

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

    PARTS:
    Or(a=in[0],b=in[1],out=c1);
    Or(a=c1,b=in[2],out=c2);
    Or(a=c2,b=in[3],out=c3);
    Or(a=c3,b=in[4],out=c4);
    Or(a=c4,b=in[5],out=c5);
    Or(a=c5,b=in[6],out=c6);
    Or(a=c6,b=in[7],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/01/Xor.hdl

/**
 * Exclusive-or gate:
```

```
 * out = not (a == b)
 */

CHIP Xor {
    IN a, b;
    OUT out;

    PARTS:
    Not(in=a, out=a0);
    Not(in=b, out=b0);
    And(a=a,b=b0,out=c0);
    And(a=b,b=a0,out=c1);
    Or(a=c0,b=c1,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/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=carry0);
    FullAdder(a=a[1],b=b[1],c=carry0,sum=out[1],carry=carry1);
    FullAdder(a=a[2],b=b[2],c=carry1,sum=out[2],carry=carry2);
    FullAdder(a=a[3],b=b[3],c=carry2,sum=out[3],carry=carry3);
    FullAdder(a=a[4],b=b[4],c=carry3,sum=out[4],carry=carry4);
    FullAdder(a=a[5],b=b[5],c=carry4,sum=out[5],carry=carry5);
    FullAdder(a=a[6],b=b[6],c=carry5,sum=out[6],carry=carry6);
    FullAdder(a=a[7],b=b[7],c=carry6,sum=out[7],carry=carry7);
    FullAdder(a=a[8],b=b[8],c=carry7,sum=out[8],carry=carry8);
    FullAdder(a=a[9],b=b[9],c=carry8,sum=out[9],carry=carry9);
    FullAdder(a=a[10],b=b[10],c=carry9,sum=out[10],carry=carry10);
    FullAdder(a=a[11],b=b[11],c=carry10,sum=out[11],carry=carry11);
    FullAdder(a=a[12],b=b[12],c=carry11,sum=out[12],carry=carry12);
    FullAdder(a=a[13],b=b[13],c=carry12,sum=out[13],carry=carry13);
    FullAdder(a=a[14],b=b[14],c=carry13,sum=out[14],carry=carry14);
    FullAdder(a=a[15],b=b[15],c=carry14,sum=out[15],carry=carry15);
}// 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[0..15]=false, sel=zx, out=outzx);
    Not16(in=outzx, out=notoutzx);
    Mux16(a=outzx, b=notoutzx, sel=nx, out=outnx);

    Mux16(a=y, b[0..15]=false, sel=zy, out=outzy);
    Not16(in=outzy, out=notoutzy);
    Mux16(a=outzy, b=notoutzy, sel=ny, out=outny);

    And16(a=outnx,b=outny,out=rand);
    Add16(a=outnx,b=outny,out=radd);

    Mux16(a=rand,b=radd,sel=f,out=outf);
```

```
    Not16(in=outf,out=notoutf);
    Mux16(a=outf, b=notoutf, sel=no,  out[0..7]=out0 , out[8..15]=out1);

    Or16(a[0..15]=false, b[0..7]=out0, b[8..15]=out1, out[0..14]=ignore, out[15]=ng);

    Or16(a[0..15]=false,b[0..7]=out0, b[8..15]=out1,out=out);


    Or8Way(in=out0,out=zr0);
    Or8Way(in=out1,out=zr1);
    Or(a=zr0,b=zr1,out=zr2);
    Not(in=zr2,out=zr);
}// 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=c,b=b,sum=sum0,carry=carry0);
    HalfAdder(a=sum0,b=a,sum=sum, carry=carry1);
    Or(a=carry0, b=carry1, 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/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/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, b[1..15]=false, 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/03/a/Bit.hdl

/**
 * 1-bit register:
 * If load[t] == 1 then out[t+1] = in[t]
 *                 else out does not change (out[t+1] = out[t])
 */

CHIP Bit {
    IN in, load;
    OUT out;

    PARTS:
    Mux(a=out1,b=in,sel=load,out=out0);
    DFF(in=out0,out=out1,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/03/a/PC.hdl

/**
 * A 16-bit counter with load and reset control bits.
 * if      (reset[t] == 1) out[t+1] = 0
 * else if (load[t] == 1)  out[t+1] = in[t]
 * else if (inc[t] == 1)   out[t+1] = out[t] + 1  (integer addition)
 * else                    out[t+1] = out[t]
 */

CHIP PC {
    IN in[16],load,inc,reset;
    OUT out[16];
```

```
    PARTS:
    Register(in=resetout,load=loadorincorreset,out=r0,out=out);
    Or(a=inc,b=load,out=loadorinc);
    Or(a=loadorinc,b=reset,out=loadorincorreset);
    // load has priority over inc
    Mux16(a=outinc,b=in,sel=load,out=inload);
    // reset has top priority
    Inc16(in=r0,out=outinc);
    Mux16(a=in,b=outinc,sel=inc,out=out0);
    Mux16(a=inload,b[0..15]=false,sel=reset,out=resetout);
}
// 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/03/a/RAM64.hdl

/**
 * Memory of 64 registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM64 {
    IN in[16], load, address[6];
    OUT out[16];

    PARTS:
    RAM8(in=in,load=load0,address=address[0..2],out=out0);
    RAM8(in=in,load=load1,address=address[0..2],out=out1);
    RAM8(in=in,load=load2,address=address[0..2],out=out2);
    RAM8(in=in,load=load3,address=address[0..2],out=out3);
    RAM8(in=in,load=load4,address=address[0..2],out=out4);
    RAM8(in=in,load=load5,address=address[0..2],out=out5);
    RAM8(in=in,load=load6,address=address[0..2],out=out6);
    RAM8(in=in,load=load7,address=address[0..2],out=out7);

DMux8Way(in=load,sel=address[3..5],a=load0,b=load1,c=load2,d=load3,e=load4,f=l
oad5,g=load6,h=load7);

Mux8Way16(a=out0,b=out1,c=out2,d=out3,e=out4,f=out5,g=out6,h=out7,sel=addres
s[3..5],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/03/a/RAM8.hdl

/**
```

```
 * Memory of 8 registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM8 {
    IN in[16], load, address[3];
    OUT out[16];

    PARTS:
    Register(in=in,load=sel0,out=out0);
    Register(in=in,load=sel1,out=out1);
    Register(in=in,load=sel2,out=out2);
    Register(in=in,load=sel3,out=out3);
    Register(in=in,load=sel4,out=out4);
    Register(in=in,load=sel5,out=out5);
    Register(in=in,load=sel6,out=out6);
    Register(in=in,load=sel7,out=out7);

DMux8Way(in=load,sel=address,a=sel0,b=sel1,c=sel2,d=sel3,e=sel4,f=sel5,g=sel6,
h=sel7);

Mux8Way16(a=out0,b=out1,c=out2,d=out3,e=out4,f=out5,g=out6,h=out7,sel=addres
s,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/03/a/Register.hdl

/**
 * 16-bit register:
 * If load[t] == 1 then out[t+1] = in[t]
 * else out does not change
 */

CHIP Register {
    IN in[16], load;
    OUT out[16];

    PARTS:
    Bit(in=in[0],load=load,out=out[0]);
    Bit(in=in[1],load=load,out=out[1]);
    Bit(in=in[2],load=load,out=out[2]);
    Bit(in=in[3],load=load,out=out[3]);
    Bit(in=in[4],load=load,out=out[4]);
    Bit(in=in[5],load=load,out=out[5]);
    Bit(in=in[6],load=load,out=out[6]);
    Bit(in=in[7],load=load,out=out[7]);
    Bit(in=in[8],load=load,out=out[8]);
```

```
    Bit(in=in[9],load=load,out=out[9]);
    Bit(in=in[10],load=load,out=out[10]);
    Bit(in=in[11],load=load,out=out[11]);
    Bit(in=in[12],load=load,out=out[12]);
    Bit(in=in[13],load=load,out=out[13]);
    Bit(in=in[14],load=load,out=out[14]);
    Bit(in=in[15],load=load,out=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/03/b/RAM16K.hdl

/**
 * Memory of 16K registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM16K {
    IN in[16], load, address[14];
    OUT out[16];

    PARTS:
    RAM4K(in=in,load=load0,address=address[0..11],out=out0);
    RAM4K(in=in,load=load1,address=address[0..11],out=out1);
    RAM4K(in=in,load=load2,address=address[0..11],out=out2);
    RAM4K(in=in,load=load3,address=address[0..11],out=out3);
    DMux4Way(in=load,sel=address[12..13],a=load0,b=load1,c=load2,d=load3);
    Mux4Way16(a=out0,b=out1,c=out2,d=out3,sel=address[12..13],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/03/b/RAM4K.hdl

/**
 * Memory of 4K registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM4K {
    IN in[16], load, address[12];
    OUT out[16];

    PARTS:
    RAM512(in=in,load=load0,address=address[0..8],out=out0);
    RAM512(in=in,load=load1,address=address[0..8],out=out1);
```

```
    RAM512(in=in,load=load2,address=address[0..8],out=out2);
    RAM512(in=in,load=load3,address=address[0..8],out=out3);
    RAM512(in=in,load=load4,address=address[0..8],out=out4);
    RAM512(in=in,load=load5,address=address[0..8],out=out5);
    RAM512(in=in,load=load6,address=address[0..8],out=out6);
    RAM512(in=in,load=load7,address=address[0..8],out=out7);

DMux8Way(in=load,sel=address[9..11],a=load0,b=load1,c=load2,d=load3,e=load4,f=
load5,g=load6,h=load7);

Mux8Way16(a=out0,b=out1,c=out2,d=out3,e=out4,f=out5,g=out6,h=out7,sel=addres
s[9..11],out=out);
}// This file is part of the materials accompanying the book
// "The Elements of Computing Systems" by Nisan and Schocken,
// MIT Press. Book site: www.idc.ac.il/tecs
// File name: projects/03/b/RAM512.hdl

/**
 * Memory of 512 registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM512 {
    IN in[16], load, address[9];
    OUT out[16];

    PARTS:
    RAM64(in=in,load=load0,address=address[0..5],out=out0);
    RAM64(in=in,load=load1,address=address[0..5],out=out1);
    RAM64(in=in,load=load2,address=address[0..5],out=out2);
    RAM64(in=in,load=load3,address=address[0..5],out=out3);
    RAM64(in=in,load=load4,address=address[0..5],out=out4);
    RAM64(in=in,load=load5,address=address[0..5],out=out5);
    RAM64(in=in,load=load6,address=address[0..5],out=out6);
    RAM64(in=in,load=load7,address=address[0..5],out=out7);

DMux8Way(in=load,sel=address[6..8],a=load0,b=load1,c=load2,d=load3,e=load4,f=l
oad5,g=load6,h=load7);

Mux8Way16(a=out0,b=out1,c=out2,d=out3,e=out4,f=out5,g=out6,h=out7,sel=addres
s[6..8],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/01/And8Way.hdl

/**
 * 8-way And:
```

```
 * out = (in[0] or in[1] or ... or in[7])
 */

CHIP And8Way {
    IN in[8];
    OUT out;

    PARTS:
    And(a=in[0],b=in[1],out=c1);
    And(a=c1,b=in[2],out=c2);
    And(a=c2,b=in[3],out=c3);
    And(a=c3,b=in[4],out=c4);
    And(a=c4,b=in[5],out=c5);
    And(a=c5,b=in[6],out=c6);
    And(a=c6,b=in[7],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/05/Computer.hdl

/**
 * The HACK computer, including CPU, ROM and RAM.
 * When reset is 0, the program stored in the computer's ROM executes.
 * When reset is 1, the execution of the program restarts.
 * Thus, to start a program's execution, reset must be pushed "up" (1)
 * and "down" (0). From this point onward the user is at the mercy of
 * the software. In particular, depending on the program's code, the
 * screen may show some output and the user may be able to interact
 * with the computer via the keyboard.
 */

CHIP Computer {

    IN reset;

    PARTS:
    ROM32K(address=RomAddress ,out=Instruction );
    CPU(inM=RamOut , instruction=Instruction , reset=reset ,outM=RamIn ,
writeM=RamLoad , addressM=RamAddress , pc=RomAddress );
    Memory(in=RamIn ,load=RamLoad , address=RamAddress , out=RamOut );
}
// 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/05/CPU.hdl

/**
 * The Hack CPU (Central Processing unit), consisting of an ALU,
 * two registers named A and D, and a program counter named PC.
 * The CPU is designed to fetch and execute instructions written in
```

```
 * the Hack machine language. In particular, functions as follows:
 * Executes the inputted instruction according to the Hack machine
 * language specification. The D and A in the language specification
 * refer to CPU-resident registers, while M refers to the external
 * memory location addressed by A, i.e. to Memory[A]. The inM input
 * holds the value of this location. If the current instruction needs
 * to write a value to M, the value is placed in outM, the address
 * of the target location is placed in the addressM output, and the
 * writeM control bit is asserted. (When writeM==0, any value may
 * appear in outM). The outM and writeM outputs are combinational:
 * they are affected instantaneously by the execution of the current
 * instruction. The addressM and pc outputs are clocked: although they
 * are affected by the execution of the current instruction, they commit
 * to their new values only in the next time step. If reset==1 then the
 * CPU jumps to address 0 (i.e. pc is set to 0 in next time step) rather
 * than to the address resulting from executing the current instruction.
 */

CHIP CPU {

    IN  inM[16],         // M value input  (M = contents of RAM[A])
        instruction[16], // Instruction for execution
        reset;           // Signals whether to re-start the current
                         // program (reset==1) or continue executing
                         // the current program (reset==0).

    OUT outM[16],        // M value output
        writeM,          // Write to M?
        addressM[15],    // Address in data memory (of M)
        pc[15];          // address of next instruction

    PARTS:

Or16(a=instruction,b[0..15]=false,out[0]=j3,out[1]=j2,out[2]=j1,out[3]=d3,out[4]=d2,ou
t[5]=d1,out[6]=c6,out[7]=c5,out[8]=c4,out[9]=c3,out[10]=c2,out[11]=c1,out[12]=a,out[
13..15]=UNUSED);
    Not(in=instruction[15],out=OpcodeA);
    Not(in=OpcodeA,out=OpcodeC);

    Mux16(a=OutALU ,b=instruction ,sel=OpcodeA ,out=InA); // instruction
    Mux16(a=OutA ,b=inM ,sel=a ,out=OutMemMux ); // Memory

ALU(x=OutD,y=OutMemMux,zx=c1,nx=c2,zy=c3,ny=c4,f=c5,no=c6,out=OutALU,out
=InD,out=outM,zr=zr,ng=ng) ; //DONE
    Or(a=d1,b=OpcodeA,out=LoadAReg);
    ARegister(in=InA ,load=LoadAReg , out=OutA, out[0..14]=addressM,
out[15]=UnusedOutRegA);//A register
    And(a=d2,b=OpcodeC,out=LoadDReg);
    DRegister(in=InD ,load=LoadDReg , out=OutD);//D register
    Or(a=zr,b=ng,out=ZrOrNg);
```

```
    Not(in=j1,out=nj1);
    Not(in=j2,out=nj2);
    Not(in=j3,out=nj3);
    Not(in=zr,out=nzr);
    Not(in=ng,out=g);
    And8Way(in[0]=nj1,in[1]=nj2,in[2]=nj3,in[3..7]=true,out=NoJump);
    And8Way(in[0]=nj1,in[1]=nj2,in[2]=j3, in[3]=nzr,in[4]=g,in[5..7]=true,out=jgt); //JGT
    And8Way(in[0]=nj1,in[1]=j2,in[2]=nj3, in[3]=zr,in[4..7]=true,out=jeq); //Jeq
    And8Way(in[0]=nj1,in[1]=j2,in[2]=j3, in[3]=g,in[4..7]=true,out=jge); //Jge
    And8Way(in[0]=j1,in[1]=nj2,in[2]=nj3, in[3]=ng,in[4..7]=true,out=jlt); //Jlt
    And8Way(in[0]=j1,in[1]=nj2,in[2]=j3, in[3]=nzr,in[4..7]=true,out=jne); //Jne
    And8Way(in[0]=j1,in[1]=j2,in[2]=nj3, in[3]=ZrOrNg, in[4..7]=true,out=jle); //Jle
    And8Way(in[0]=j1,in[1]=j2,in[2]=j3,in[3..7]=true,out=jmp); //Jmp


Or8Way(in[0]=jgt,in[1]=jeq,in[2]=jge,in[3]=jlt,in[4]=jne,in[5]=jle,in[6]=jmp,in[7]=false,o
ut=LoadPC0);
    And(a=OpcodeC,b=LoadPC0,out=LoadPC);

    Not(in=LoadPC,out=Inc);
    PC(in=OutA ,load=LoadPC ,inc=Inc, reset=reset, out[15]=false, out[0..14]=pc);//
PC register
    And(a=d3,b=OpcodeC,out=writeM);
}
// 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/05/Memory.hdl

/**
 * The complete address space of the Hack computer's memory,
 * including RAM and memory-mapped I/O.
 * The chip facilitates read and write operations, as follows:
 *     Read:  out(t) = Memory[address(t)](t)
 *     Write: if load(t-1) then Memory[address(t-1)](t) = in(t-1)
 * In words: the chip always outputs the value stored at the memory
 * location specified by address. If load==1, the in value is loaded
 * into the memory location specified by address. This value becomes
 * available through the out output from the next time step onward.
 * Address space rules:
 * Only the upper 16K+8K+1 words of the Memory chip are used.
 * Access to address>0x6000 is invalid. Access to any address in
 * the range 0x4000-0x5FFF results in accessing the screen memory
 * map. Access to address 0x6000 results in accessing the keyboard
 * memory map. The behavior in these addresses is described in the
 * Screen and Keyboard chip specifications given in the book.
 */

CHIP Memory {
    IN in[16], load, address[15];
```

```
    OUT out[16];

    PARTS:
    RAM16K(in=in,load=load1,address=address[0..13],out=out1);
    Screen(in=in,load=load2,address=address[0..12],out=out2);
    Keyboard(out=out3);
    DMux4Way(in=load,sel=address[13..14],a=load1a,b=load1b,c=load2,d=load3);
    Or(a=load1a,b=load1b,out=load1);
    Mux4Way16(a=out1,b=out1,c=out2,d=out3,sel=address[13..14],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/demo/Xor.hdl

/**
 *  Exclusive-or gate: true if either a is true and b is false, or
 *  a is false and b is true; false otherwise.
 *  QUESTION: how can the simulator execute this program properly without
 *  HDL implementations of the underlying Not, And, and Or chip-parts?
 *  Answer: since the demo folder contains no Not.hdl, And.hdl and Or.hdl
 *  files, the simulator reverts to using their built-in implementations.
 */

CHIP Xor {
    IN a, b;
    OUT out;

    PARTS:
    Not (in=a, out=nota);
    Not (in=b, out=notb);
    And (a=a, b=notb, out=x);
    And (a=nota, b=b, out=y);
    Or (a=x, b=y, out=out);
}
```