COMP1036 Computer Fundamentals
Lab 4

Implement the following circuits:

1. HalfAdder

```
Half adder. Computes sum, the least significant bit of a + b,
and carry, the most significant bit of a + b.
```

Solution

```
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);
}
```

2. FullAdder

```
Full adder. Computes sum, the least significant bit of
a + b + c, and carry, the most significant bit of a + b + c.
```

Solution

```
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=w1, carry=c1);
HalfAdder(a=w1, b=c, sum=sum, carry=c2);
Xor(a=c1, b=c2, out=carry);
}
```

3. Add16

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

Solution

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

    PARTS:
    HalfAdder(a=a[0], b=b[0], sum=out[0], carry=carry1);
    FullAdder(a=a[1], b=b[1], c=carry1, sum=out[1], carry=carry2);
    FullAdder(a=a[2], b=b[2], c=carry2, sum=out[2], carry=carry3);
    FullAdder(a=a[3], b=b[3], c=carry3, sum=out[3], carry=carry4);
    FullAdder(a=a[4], b=b[4], c=carry4, sum=out[4], carry=carry5);
    FullAdder(a=a[5], b=b[5], c=carry5, sum=out[5], carry=carry6);
    FullAdder(a=a[6], b=b[6], c=carry6, sum=out[6], carry=carry7);
    FullAdder(a=a[7], b=b[7], c=carry7, sum=out[7], carry=carry8);
    FullAdder(a=a[8], b=b[8], c=carry8, sum=out[8], carry=carry9);
    FullAdder(a=a[9], b=b[9], c=carry9, sum=out[9], carry=carry10);
    FullAdder(a=a[10], b=b[10], c=carry10, sum=out[10], carry=carry11);
    FullAdder(a=a[11], b=b[11], c=carry11, sum=out[11], carry=carry12);
    FullAdder(a=a[12], b=b[12], c=carry12, sum=out[12], carry=carry13);
    FullAdder(a=a[13], b=b[13], c=carry13, sum=out[13], carry=carry14);
    FullAdder(a=a[14], b=b[14], c=carry14, sum=out[14], carry=carry15);
    FullAdder(a=a[15], b=b[15], c=carry15, sum=out[15], carry=carry16);
}
```

4. Inc16

```
    16-bit incrementer.  out = in + 1 (16-bit addition).
    Ignore the overflow.
```

Solution

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

    PARTS:
    Add16(a[0..15]=in[0..15], b[0]=true, b[1..15]=false, out[0..15]=out[0..15]);
}
```

5. PC

```
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]
```

Solution

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

PARTS:
// increment the output of the register
Inc16(in = feedback, out = pc);
Mux16(a = feedback, b = pc, sel = inc, out = w0);
Mux16(a = w0, b = in, sel = load, out = w1);
Mux16(a = w1, b = false, sel = reset, out = cout);
// the output from the register also needs to get fed back through
// the combinational logic to get processed for the next clock cycle.
Register(in = cout, load = true, out = out, out = feedback);
}
```