Contents

1 Basic Test Results

```
********** TESTING FOLDER STRUCTURE START **********
Checking your submission for presence of invalid (non-ASCII) characters...
No invalid characters found.
Submission logins are: linorcohen
Is this OK?
**********  TESTING FOLDER STRUCTURE END  **********

********** PROJECT TEST START **********
Testing.
Add16 passed test.
FullAdder passed test.
HalfAdder passed test.
Inc16 passed test.
**********  PROJECT TEST END  **********

Note: the tests you see above are all the presubmission tests
for this project. The tests might not check all the different
parts of the project or all corner cases, so write your own
tests and use them!
```

2 ALU.hdl

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).
// 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:
    // You're advised to work on the ALU chip in two steps:
    // - First, without handling status outputs (ALU-nostat)
    // - Then, adding the missing functionality for the "full" chip (ALU).
    // You only need to submit the "full" ALU, no need to submit the partial
    // implementation (ALU-nostat).
    // Put your code here:

    Mux16(a=y, b=false, sel=zy, out=zyres);
    Not16(in=zyres, out=yNot);
    Mux16(a=zyres, b=yNot, sel=ny, out=nyres);
    And16(a=nyres, b=nxres, out=nyresAndnxres);

    Mux16(a=x, b=false, sel=zx, out=zxres);
```

```
    Not16(in=zxres, out=xNot);
    Mux16(a=zxres, b=xNot, sel=nx, out=nxres);
    Add16(a=nyres, b=nxres, out=nyresAddnxres);

    Mux16(a=nyresAndnxres, b=nyresAddnxres, sel=f, out=fres);
    Not16(in=fres, out=fresNot);
    Mux16(a=fres, b=fresNot, sel=no, out[0..7]=subout1, out[8..15]=subout2, out=out);

    Or8Way(in=subout1, out=subout1Or);
    Or8Way(in=subout2, out=subout2Or);
    Or(a=subout1Or, b=subout2Or, out=sub1Orsub2);
    Not(in=sub1Orsub2, out=zr);

    And16(a[0..7]=subout1, a[8..15]=subout2, b=true, out[15]=ng);
}
```

3 AUTHORS

```
linorcohen
Partner 1: Linor Cohen, linor.cohen@mail.huji.ac.il, 318861226
Remarks:
```

4 Add16.hdl

```
// 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=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=carry);
}
```

5 FullAdder.hdl

```
// 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=sum1, carry=carry1);
    HalfAdder(a=sum1, b=c, sum=sum, carry=carry2);
    Or(a=carry1, b=carry2, out=carry);
}
```

6 HalfAdder.hdl

```
// 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);
}
```

7 Inc16.hdl

```
// 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[1..15]=false, b[0]=true, out=out);
}
```

8 ShiftLeft.hdl

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

/**
 * 16-bit left shifter.
 *
 * The chip's output is a "left-shift" of the input:
 * - Every input bit is moved one position to the left
 * - A new "0" bit is inserted as the new right-most bit
 *
 * For example:
 * ShiftLeft(0000000000000001)=0000000000000010 // ShiftLeft(1)=2
 * ShiftLeft(0100000000000000)=1000000000000000
 * ShiftLeft(1000000000000000)=0000000000000000
 *
 * This operation is (usually) equivalent to multiplying the input by 2.
 * This definition is also called an arithmetic left-shift, and is useful for
 * the efficient implementation of various operations which we will see later on
 * in the course.
 */

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

    PARTS:
    Add16(a=in, b=in, out=out);
}
```

9 ShiftRight.hdl

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

/**
 * 16-bit right-shifter.
 *
 * The chip's output is a "right-shift" of the input:
 * - Every input bit is moved one position to the right
 * - A new bit which is equal to the sign bit is inserted as the left-most bit
 *
 * For example:
 * ShiftRight(0000000000000001)=0000000000000000 // ShiftRight(1)=0
 * ShiftRight(0100000000000000)=0010000000000000
 * ShiftRight(1100000000000000)=1110000000000000
 *
 * Note that this operation is (usually) equivalent to dividing the input by 2.
 * This definition is also called an arithmetic right-shift, and is useful for
 * the efficient implementation of various operations which we will see later on
 * in the course.
 * Another variant is the logical right-shift, which always inserts a new '0'
 * bit.
 */

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

    PARTS:
    Or16(a[0..14]=in[1..15], a[15]=in[15], b=false, out=out);
}
```