Contents

1 Basic Test Results

```
********** TESTING FOLDER STRUCTURE START **********
Running test2.sh:
Your logins are: muaz.abdeen, is that ok?

********** TESTING FOLDER STRUCTURE END **********

********** PROJECT TEST START **********

FullAdder passed test
HalfAdder passed test

********** PROJECT TEST END **********
```

2 README

```
muaz.abdeen
================================================================================
Muaz Abdeen, ID 300575297, muaz.abdeen@mail.huji.ac.il
================================================================================

                           Project 2 - Boolean Arithmetic
                           ------------------------------


Submitted Files
---------------
(1)  README             - This file.
(2)  HalfAdder.hdl        - The HalfAdder gate implementation.
(3)  FullAdder.hdl        - The FullfAdder gate implementation.
(4)  Add16.hdl            - The Add16 gate implementation.
(5)  Inc16.hdl            - The Inc16 gate implementation.
(6)  ALU.hdl            - The implementation of the ALU.
(7)  ShiftLeft.hdl        - The ShiftLeft gate implementation.
(8)  ShiftRight.hdl        - The ShiftRight gate implementation.
(9)  Mul.hdl            - The Mul gate implementation.
(10) LogicalShiftLeft.hdl    - A supplementary chip in order to ease the building of the
                  the multiplication chip (Mul.hdl).
                  It is a 16-bit value logical left shift.
                   It moves every input bit one position to the left, include
                  the sign bit (the sign might change).
                   The "new" right-most bit should be 0.



Remarks
-------
* In HalfAdder.hdl I reimplement Xor gate, in order to reuse the first Nand gate in the Not gate.
  In the less efficient implementation we can use Xor and And gates which costs (6 Nands), so I negated
  the first Nand gate, and saved a Nand gate (And == 2 Nands).
* In FullAdder.hdl it is more efficient if not reusing Half Adder, but using Nands only (9 Nands).
* In Inc16.hdl I exploit the advantage of automatic size fitting of a multi-bit bus.
```

3 ALU.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/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:
    // zero or negate the x input?
    Mux16(a=x ,b=false ,sel=zx ,out=zxRes);
    Not16(in=zxRes ,out=NotzxRes);
    Mux16(a=zxRes ,b=NotzxRes ,sel=nx ,out=nxRes);

    // zero or negate the y input?
    Mux16(a=y ,b=false ,sel=zy ,out=zyRes);
    Not16(in=zyRes ,out=NotzyRes);
    Mux16(a=zyRes ,b=NotzyRes ,sel=ny ,out=nyRes);

    // determine the function (And, Add)?
    And16(a=nxRes ,b=nyRes ,out=xAndy);
    Add16(a=nxRes ,b=nyRes ,out=xAddy);
    Mux16(a=xAndy ,b=xAddy ,sel=f ,out=fRes);

```

```
60      // negate the out output? and determine if (out < 0)
61      Not16(in=fRes ,out=NotfRes);
62      Mux16(a=fRes ,b=NotfRes ,sel=no ,out=out ,out[15]=ng ,out[0..7]=temp1 ,out[8..15]=temp2);
63
64      // determine if (out == 0)
65      Or8Way(in=temp1 ,out=zrRes1);
66      Or8Way(in=temp2 ,out=zrRes2);
67      Or(a=zrRes1 ,b=zrRes2, out=Notzr);
68      Not(in=Notzr ,out=zr);
69  }
```

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=carr0);
    FullAdder(a=a[1] ,b=b[1] ,c=carr0 ,sum=out[1], carry=carr1);
    FullAdder(a=a[2] ,b=b[2] ,c=carr1 ,sum=out[2], carry=carr2);
    FullAdder(a=a[3] ,b=b[3] ,c=carr2 ,sum=out[3], carry=carr3);
    FullAdder(a=a[4] ,b=b[4] ,c=carr3 ,sum=out[4], carry=carr4);
    FullAdder(a=a[5] ,b=b[5] ,c=carr4 ,sum=out[5], carry=carr5);
    FullAdder(a=a[6] ,b=b[6] ,c=carr5 ,sum=out[6], carry=carr6);
    FullAdder(a=a[7] ,b=b[7] ,c=carr6 ,sum=out[7], carry=carr7);
    FullAdder(a=a[8] ,b=b[8] ,c=carr7 ,sum=out[8], carry=carr8);
    FullAdder(a=a[9] ,b=b[9] ,c=carr8 ,sum=out[9], carry=carr9);
    FullAdder(a=a[10] ,b=b[10] ,c=carr9 ,sum=out[10], carry=carr10);
    FullAdder(a=a[11] ,b=b[11] ,c=carr10 ,sum=out[11], carry=carr11);
    FullAdder(a=a[12] ,b=b[12] ,c=carr11 ,sum=out[12], carry=carr12);
    FullAdder(a=a[13] ,b=b[13] ,c=carr12 ,sum=out[13], carry=carr13);
    FullAdder(a=a[14] ,b=b[14] ,c=carr13 ,sum=out[14], carry=carr14);
    FullAdder(a=a[15] ,b=b[15] ,c=carr14 ,sum=out[15], carry=carr15);
}
```

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:
    // more efficient by not reusing Half Adder, but Nands only (9 Nands)
    Nand(a=a ,b=b ,out=aNandb);
    Nand(a=a ,b=aNandb ,out=res1);
    Nand(a=aNandb ,b=b ,out=res2);
    Nand(a=res1 ,b=res2 ,out=aXorb);
    Nand(a=aXorb ,b=c ,out=cNandaXorb);
    Nand(a=aXorb ,b=cNandaXorb ,out=res3);
    Nand(a=cNandaXorb ,b=c ,out=res4);
    Nand(a=res3 ,b=res4 ,out=sum);
    Nand(a=aNandb ,b=cNandaXorb ,out=carry);


    // Less efficient: (5+5+3=13 Nands)
    // HalfAdder(a=a ,b=b ,sum=partSum ,carry=Partcarry1);
    // HalfAdder(a=partSum ,b=c ,sum=sum ,carry=Partcarry2);
    // Or(a=Partcarry1 ,b=Partcarry2 ,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:
    // re-implement Xor gate, in order to re-use the first Nand gate in the Not gate.
    Nand(a=a ,b=b ,out=aNandb);
    Nand(a=aNandb ,b=a ,out=res1);
    Nand(a=aNandb ,b=b ,out=res2);
    Nand(a=res1 ,b=res2 ,out=sum);
    Not(in=aNandb, out=carry);

    // Less efficient implementaion (6 Nands).
    // 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[0]=true, out=out);
}
```

8 LogicalShiftLeft.hdl

```
1
2  // File name: projects/02/LogicalShiftLeft.hdl
3
4  /**
5   * 16-bit value logical left shift.
6   * The chip should move every input bit one position to
7   * the left, include the sign bit (the sign might change).
8   * The "new" right-most bit should be 0.
9   */
10
11 CHIP LogicalShiftLeft{
12      IN in[16];
13      OUT out[16];
14
15      PARTS:
16      Nand(a=true ,b=true ,out=out[0]);
17      And(a=in[0] ,b=true ,out=out[1]);
18      And(a=in[1] ,b=true ,out=out[2]);
19      And(a=in[2] ,b=true ,out=out[3]);
20      And(a=in[3] ,b=true ,out=out[4]);
21      And(a=in[4] ,b=true ,out=out[5]);
22      And(a=in[5] ,b=true ,out=out[6]);
23      And(a=in[6] ,b=true ,out=out[7]);
24      And(a=in[7] ,b=true ,out=out[8]);
25      And(a=in[8] ,b=true ,out=out[9]);
26      And(a=in[9] ,b=true ,out=out[10]);
27      And(a=in[10] ,b=true ,out=out[11]);
28      And(a=in[11] ,b=true ,out=out[12]);
29      And(a=in[12] ,b=true ,out=out[13]);
30      And(a=in[13] ,b=true ,out=out[14]);
31      And(a=in[14] ,b=true ,out=out[15]);
32 }
```

9 Mul.hdl

```
// This file is part of course: "Workshop In Computer
// Construction: From Nand to Tetris" by HUJI.
// File name: projects/02/Mul.hdl

/**
 * This chip multiplies 2 numbers.
 * Handling overflows: any number larger than 16 bits can
 * be truncated to include only the 16 least significant bits.
 */

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

    PARTS:

    // calculating the partial products (16 products like number of bits)
    Mux16(a=false ,b=a ,sel=b[0] ,out=part0);           // 1st partial product
    LogicalShiftLeft(in=a ,out=shift1);
    Mux16(a=false ,b=shift1 ,sel=b[1] ,out=part1);      // 2nd partial product
    LogicalShiftLeft(in=shift1 ,out=shift2);
    Mux16(a=false ,b=shift2 ,sel=b[2] ,out=part2);      // 3rd partial product
    LogicalShiftLeft(in=shift2 ,out=shift3);
    Mux16(a=false ,b=shift3 ,sel=b[3] ,out=part3);      // 4th partial product
    LogicalShiftLeft(in=shift3 ,out=shift4);
    Mux16(a=false ,b=shift4 ,sel=b[4] ,out=part4);      // 5th partial product
    LogicalShiftLeft(in=shift4 ,out=shift5);
    Mux16(a=false ,b=shift5 ,sel=b[5] ,out=part5);      // 6th partial product
    LogicalShiftLeft(in=shift5 ,out=shift6);
    Mux16(a=false ,b=shift6 ,sel=b[6] ,out=part6);      // 7th partial product
    LogicalShiftLeft(in=shift6 ,out=shift7);
    Mux16(a=false ,b=shift7 ,sel=b[7] ,out=part7);      // 8th partial product
    LogicalShiftLeft(in=shift7 ,out=shift8);
    Mux16(a=false ,b=shift8 ,sel=b[8] ,out=part8);      // 9th partial product
    LogicalShiftLeft(in=shift8 ,out=shift9);
    Mux16(a=false ,b=shift9 ,sel=b[9] ,out=part9);      // 10th partial product
    LogicalShiftLeft(in=shift9 ,out=shift10);
    Mux16(a=false ,b=shift10 ,sel=b[10] ,out=part10);   // 11th partial product
    LogicalShiftLeft(in=shift10 ,out=shift11);
    Mux16(a=false ,b=shift11 ,sel=b[11] ,out=part11);   // 12th partial product
    LogicalShiftLeft(in=shift11 ,out=shift12);
    Mux16(a=false ,b=shift12 ,sel=b[12] ,out=part12);   // 13th partial product
    LogicalShiftLeft(in=shift12 ,out=shift13);
    Mux16(a=false ,b=shift13 ,sel=b[13] ,out=part13);   // 14th partial product
    LogicalShiftLeft(in=shift13 ,out=shift14);
    Mux16(a=false ,b=shift14 ,sel=b[14] ,out=part14);   // 15th partial product
    LogicalShiftLeft(in=shift14 ,out=shift15);
    Mux16(a=false ,b=shift15 ,sel=b[15] ,out=part15);   // 16th partial product

    // Sum the patrial products
    Add16(a=part0 ,b=part1, out=res0);
    Add16(a=res0 ,b=part2, out=res1);
    Add16(a=res1 ,b=part3, out=res2);
    Add16(a=res2 ,b=part4, out=res3);
    Add16(a=res3 ,b=part5, out=res4);
    Add16(a=res4 ,b=part6, out=res5);
    Add16(a=res5 ,b=part7, out=res6);
    Add16(a=res6 ,b=part8, out=res7);
    Add16(a=res7 ,b=part9, out=res8);
```

```
60      Add16(a=res8 ,b=part10, out=res9);
61      Add16(a=res9 ,b=part11, out=res10);
62      Add16(a=res10 ,b=part12, out=res11);
63      Add16(a=res11 ,b=part13, out=res12);
64      Add16(a=res12 ,b=part14, out=res13);
65      Add16(a=res13 ,b=part15, out=out);
66
67  }
```

10 ShiftLeft.hdl

```
// This file is part of course: "Workshop In Computer
// Construction: From Nand to Tetris" by HUJI.
// File name: projects/02/ShiftLeft.hdl

/**
 * 16-bit value arithmetic left shift.
 * The chip should move every input bit one position to
 * the left, except the sign bit (the sign should not change).
 * The "new" right-most bit should be 0.
 */

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

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

11 ShiftRight.hdl

```
// This file is part of course: "Workshop In Computer
// Construction: From Nand to Tetris" by HUJI.
// File name: projects/02/ShiftRight.hdl

/**
 * 16-bit value arithmetic right shift.
 * The chip should move every input bit one position to
 * the right, except the sign bit (the sign should not change).
 * The "new" bit should be the same as the sign.
 */

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

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

}
```