Contents

1 README

```
nivkeren,ransha
================================================================================
Niv Keren, ID 201478351, niv.keren@cs.huji.ac.il
Ran Shaham, ID 203781000, ran.shaham1@mail.huji.ac.il
================================================================================

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


Submitted Files
---------------
README - This file.
HalfAdder.hdl - Adds two bits and outputs the sum and carry
FullAdder.hdl - Adds three bits and outputs the sum and carry
Add16.hdl - Adds two 16-bit integer numbers
Inc16.hdl - Increments a 16-bit number
ALU.hdl - The Arithmetic Logic Unit
ShiftLeft.hdl - Multiplies a 16-bit input number by 2 (ignoring overflow)
ShiftRight.hdl - Devides a 16-bit input number by 2 without remainder
Mul.hdl - Multiplies two 16-bit numbers.

Remarks
-------
* The multiplication chip was implemented according to long-multiplication algorithm
  where multiplication of each two bits x,y is just the And(x,y) operation.
  This method's shortcoming is the negative overflowing results which turns very small
  negative numbers to positive.
  Nonetheless, this method works well on the supplied test and in the exercise forum
  it is clearly written that it is sufficient.
* All other chips had simple implementations that were drawn directly from the lectures
  and simple logic.
```

2 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:
        // Get negated inputs
        Not16(in=x, out=negx);
        Not16(in=y, out=negy);

        // Choose how to process x,y according to zx,nx,zy,ny.
        Mux4Way16(a=x, b=false, c=negx, d=true, sel[0]=zx, sel[1]=nx, out=procx);
        Mux4Way16(a=y, b=false, c=negy, d=true, sel[0]=zy, sel[1]=ny, out=procy);

        // Get the correct operation of the processed x,y according to f
        And16(a=procx, b=procy, out=xAndy);
        Add16(a=procx, b=procy, out=xAddy);
        Mux16(a=xAndy, b=xAddy, sel=f, out=outf);

        // Negate output if needed and get the status outputs
        Not16(in=outf, out=negout);
```

```
60          Mux16(a=outf, b=negout, sel=no, out=out, out[15]=ng, out[0..7]=out07, out[8..15]=out815);
61
62          // Check if one of the output bits is 1, if so, the output is not zero.
63          Or8Way(in=out07, out=negzr1);
64          Or8Way(in=out815, out=negzr2);
65          Or(a=negzr1, b=negzr2, out=negzr);
66          Not(in=negzr, out=zr);
67  }
```

3 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=c1);
        FullAdder(a=a[1], b=b[1], c=c1, sum=out[1], carry=c2);
        FullAdder(a=a[2], b=b[2], c=c2, sum=out[2], carry=c3);
        FullAdder(a=a[3], b=b[3], c=c3, sum=out[3], carry=c4);
        FullAdder(a=a[4], b=b[4], c=c4, sum=out[4], carry=c5);
        FullAdder(a=a[5], b=b[5], c=c5, sum=out[5], carry=c6);
        FullAdder(a=a[6], b=b[6], c=c6, sum=out[6], carry=c7);
        FullAdder(a=a[7], b=b[7], c=c7, sum=out[7], carry=c8);
        FullAdder(a=a[8], b=b[8], c=c8, sum=out[8], carry=c9);
        FullAdder(a=a[9], b=b[9], c=c9, sum=out[9], carry=c10);
        FullAdder(a=a[10], b=b[10], c=c10, sum=out[10], carry=c11);
        FullAdder(a=a[11], b=b[11], c=c11, sum=out[11], carry=c12);
        FullAdder(a=a[12], b=b[12], c=c12, sum=out[12], carry=c13);
        FullAdder(a=a[13], b=b[13], c=c13, sum=out[13], carry=c14);
        FullAdder(a=a[14], b=b[14], c=c14, sum=out[14], carry=c15);
        FullAdder(a=a[15], b=b[15], c=c15, sum=out[15], carry=c16);
}
```

4 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:
        // Add two bits
        HalfAdder(a=a, b=b, sum=s1, carry=c1);
        // Add the third bit to the result
        HalfAdder(a=s1, b=c, sum=sum, carry=c2);
        // If one of the above additions had a carry,
        // so does the whole sum
        Or(a=c1, b=c2, out=carry);
}
```

5 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:
        // Simple half-adder implementation according to the truth-table.
        Xor(a=a, b=b, out=sum);
        And(a=a, b=b, out=carry);
}
```

6 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:
        // b = 00...01 = 1 ==> a + b = a + 1
        Add16(a=in, b[0]=true, b[1..15]=false, out=out);
}
```

7 Mul.hdl

```
/**
 * Multiplies two signed 16-bit numbers.
 */
CHIP Mul{
    IN a[16], b[16];
    OUT out[16];

    PARTS:
        // Performs binary-long-multiplication, similar to decimal multiplication

        // Get all shifted versions of 'a'
        ShiftLeft(in=a, out=sh1a);
        ShiftLeft(in=sh1a, out=sh2a);
        ShiftLeft(in=sh2a, out=sh3a);
        ShiftLeft(in=sh3a, out=sh4a);
        ShiftLeft(in=sh4a, out=sh5a);
        ShiftLeft(in=sh5a, out=sh6a);
        ShiftLeft(in=sh6a, out=sh7a);
        ShiftLeft(in=sh7a, out=sh8a);
        ShiftLeft(in=sh8a, out=sh9a);
        ShiftLeft(in=sh9a, out=sh10a);
        ShiftLeft(in=sh10a, out=sh11a);
        ShiftLeft(in=sh11a, out=sh12a);
        ShiftLeft(in=sh12a, out=sh13a);
        ShiftLeft(in=sh13a, out=sh14a);

        // The 'multiplication' operation
        And16(a=a, b[0]=b[0], b[1]=b[0], b[2]=b[0], b[3]=b[0], b[4]=b[0], b[5]=b[0], b[6]=b[0], b[7]=b[0], b[8]=b[0], b[9]=b
        And16(a=sh1a, b[0]=b[1], b[1]=b[1], b[2]=b[1], b[3]=b[1], b[4]=b[1], b[5]=b[1], b[6]=b[1], b[7]=b[1], b[8]=b[1], b[9
        And16(a=sh2a, b[0]=b[2], b[1]=b[2], b[2]=b[2], b[3]=b[2], b[4]=b[2], b[5]=b[2], b[6]=b[2], b[7]=b[2], b[8]=b[2], b[9
        And16(a=sh3a, b[0]=b[3], b[1]=b[3], b[2]=b[3], b[3]=b[3], b[4]=b[3], b[5]=b[3], b[6]=b[3], b[7]=b[3], b[8]=b[3], b[9
        And16(a=sh4a, b[0]=b[4], b[1]=b[4], b[2]=b[4], b[3]=b[4], b[4]=b[4], b[5]=b[4], b[6]=b[4], b[7]=b[4], b[8]=b[4], b[9
        And16(a=sh5a, b[0]=b[5], b[1]=b[5], b[2]=b[5], b[3]=b[5], b[4]=b[5], b[5]=b[5], b[6]=b[5], b[7]=b[5], b[8]=b[5], b[9
        And16(a=sh6a, b[0]=b[6], b[1]=b[6], b[2]=b[6], b[3]=b[6], b[4]=b[6], b[5]=b[6], b[6]=b[6], b[7]=b[6], b[8]=b[6], b[9
        And16(a=sh7a, b[0]=b[7], b[1]=b[7], b[2]=b[7], b[3]=b[7], b[4]=b[7], b[5]=b[7], b[6]=b[7], b[7]=b[7], b[8]=b[7], b[9
        And16(a=sh8a, b[0]=b[8], b[1]=b[8], b[2]=b[8], b[3]=b[8], b[4]=b[8], b[5]=b[8], b[6]=b[8], b[7]=b[8], b[8]=b[8], b[9
        And16(a=sh9a, b[0]=b[9], b[1]=b[9], b[2]=b[9], b[3]=b[9], b[4]=b[9], b[5]=b[9], b[6]=b[9], b[7]=b[9], b[8]=b[9], b[9
        And16(a=sh10a, b[0]=b[10], b[1]=b[10], b[2]=b[10], b[3]=b[10], b[4]=b[10], b[5]=b[10], b[6]=b[10], b[7]=b[10], b[8]=
        And16(a=sh11a, b[0]=b[11], b[1]=b[11], b[2]=b[11], b[3]=b[11], b[4]=b[11], b[5]=b[11], b[6]=b[11], b[7]=b[11], b[8]=
        And16(a=sh12a, b[0]=b[12], b[1]=b[12], b[2]=b[12], b[3]=b[12], b[4]=b[12], b[5]=b[12], b[6]=b[12], b[7]=b[12], b[8]=
        And16(a=sh13a, b[0]=b[13], b[1]=b[13], b[2]=b[13], b[3]=b[13], b[4]=b[13], b[5]=b[13], b[6]=b[13], b[7]=b[13], b[8]=
        And16(a=sh14a, b[0]=b[14], b[1]=b[14], b[2]=b[14], b[3]=b[14], b[4]=b[14], b[5]=b[14], b[6]=b[14], b[7]=b[14], b[8]=

        // Sum all prodcuts
        Add16(a=o0, b=o1, out=a1);
        Add16(a=a1, b=o2, out=a2);
        Add16(a=a2, b=o3, out=a3);
        Add16(a=a3, b=o4, out=a4);
        Add16(a=a4, b=o5, out=a5);
        Add16(a=a5, b=o6, out=a6);
        Add16(a=a6, b=o7, out=a7);
        Add16(a=a7, b=o8, out=a8);
        Add16(a=a8, b=o9, out=a9);
        Add16(a=a9, b=o10, out=a10);
        Add16(a=a10, b=o11, out=a11);
        Add16(a=a11, b=o12, out=a12);
        Add16(a=a12, b=o13, out=a13);
        Add16(a=a13, b=o14, out=out);
}
```

8 ShiftLeft.hdl

```
/**
 * Multiplies a 16-bit number by 2 by shifthing each bit (except the MSB) left.
 */
CHIP ShiftLeft {
     IN in[16];
     OUT out[16];

     PARTS:
        And16(a[1..15]=true, a[0]=false, b[0]=false, b[1..14]=in[0..13], b[15]=in[15], out=out);
}
```

9 ShiftRight.hdl

```
/**
 * Divides a 16-bit number by 2 by shifthing each bit (except the MSB) right.
 */
CHIP ShiftRight {
     IN in[16];
     OUT out[16];

     PARTS:
        And16(a=true, b[0..13]=in[1..14], b[14]=in[15], b[15]=in[15], out=out);
}
```