Contents

1 README

```
roigreenberg,inbaravni

===============================================================================

Roi Greenberg, ID 30557123, roi.greenberg@mail.huji.ac.il
Inbar Avni, ID 201131760, inbar.avni@mail.huji.ac.il

===============================================================================



                           Project 2 - Combinational Chips

                           -------------------------------






Submitted Files

---------------

README - This file.

HalfAdder.hdl - The HalfAdder chip.
FullAdder.hdl - The FullAdder chip.
Add16.hdl - 16-bit Adder chip.
Inc16.hdl - 16-bit Incrementer chip.
ALU.hdl - Arithmetic Logic Unit chip.
ShiftLeft.hdl - A chip that multiply by 2 its input (the sign should not change).
ShiftRight.hdl - A chip that divide by 2 its input (the sign should not change).
Mul.hdl - A chip that multiply 2 numbers.






Remarks

-------

* No remarks for that time.
```

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:
    // zero the x input if needed
    Mux16(a = x, b = false, sel = zx, out = afterZx);

    // not the x input if needed
    Not16(in = afterZx, out = afterNx);

    // choose between the afterZx and afterNx
    Mux16(a = afterZx, b = afterNx, sel = nx, out = xAfterAll);

    // zero the y input if needed
    Mux16(a = y, b = false, sel = zy, out = afterZy);

    // not the y input if needed
    Not16(in = afterZy, out = afterNy);

    // choose between the afterZy and afterNy
    Mux16(a = afterZy, b = afterNy, sel = ny, out = yAfterAll);

    // do Add if needed
    Add16(a = xAfterAll, b = yAfterAll, out = outAdd);

    // do And if needed
    And16(a = xAfterAll, b = yAfterAll, out = outAnd);

    // choose between Add and And
    Mux16(a = outAnd, b = outAdd, sel = f, out = middleOut);

    // do not on output
    Not16(in = middleOut, out = notMiddleOut);

    // take the not-output value if needed
    Mux16(a = middleOut, b = notMiddleOut, sel = no, out[0..7] = out07, out[8..14] = out814, out[15] = out15);

    // determine if negative output
    And(a = out15, b = true, out = ng);

    // determine if output is zero
    Or8Way(in = out07, out = or1);
    Or8Way(in[0..6] = out814, in[7] = out15, out = or2);
    Or(a = or1, b = or2, out = or);
    Not(in = or, out = zr);

    // the out
    And16(a[0..7] = out07, a[8..14] = out814, a[15] = out15, b = true, out = out);
}
```

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 = middlecarry0);
    FullAdder(a = a[1], b = b[1], c = middlecarry0,  sum = out[1], carry = middlecarry1);
    FullAdder(a = a[2], b = b[2], c = middlecarry1,  sum = out[2], carry = middlecarry2);
    FullAdder(a = a[3], b = b[3], c = middlecarry2,  sum = out[3], carry = middlecarry3);
    FullAdder(a = a[4], b = b[4], c = middlecarry3,  sum = out[4], carry = middlecarry4);
    FullAdder(a = a[5], b = b[5], c = middlecarry4,  sum = out[5], carry = middlecarry5);
    FullAdder(a = a[6], b = b[6], c = middlecarry5,  sum = out[6], carry = middlecarry6);
    FullAdder(a = a[7], b = b[7], c = middlecarry6,  sum = out[7], carry = middlecarry7);
    FullAdder(a = a[8], b = b[8], c = middlecarry7,  sum = out[8], carry = middlecarry8);
    FullAdder(a = a[9], b = b[9], c = middlecarry8,  sum = out[9], carry = middlecarry9);
    FullAdder(a = a[10], b = b[10], c = middlecarry9,  sum = out[10], carry = middlecarry10);
    FullAdder(a = a[11], b = b[11], c = middlecarry10,  sum = out[11], carry = middlecarry11);
    FullAdder(a = a[12], b = b[12], c = middlecarry11,  sum = out[12], carry = middlecarry12);
    FullAdder(a = a[13], b = b[13], c = middlecarry12,  sum = out[13], carry = middlecarry13);
    FullAdder(a = a[14], b = b[14], c = middlecarry13,  sum = out[14], carry = middlecarry14);
    FullAdder(a = a[15], b = b[15], c = middlecarry14,  sum = out[15], carry = ignoredBit);
}
```

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:
    HalfAdder(a = a, b = b, sum = middleSum, carry = carry1);
    HalfAdder(a = middleSum, b = c, sum = sum, carry = carry2);
    Or(a = carry1, b = carry2, 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:
    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:
   Add16(a = in, b[0] = true, b[1..15] = false, out = out);
}
```

7 Mul.hdl

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

    PARTS:
    Mux16(a[0..14] = false, b = a, sel = b[0], out = mul0);
    Add16(a = false, b = mul0, out = add0);
    ShiftLeft(in = a, out = shift1);

    Mux16(a[0..14] = false, b = shift1, sel = b[1], out = mul1);
    Add16(a = add0, b = mul1, out = add1);
    ShiftLeft(in = shift1, out = shift2);

    Mux16(a[0..14] = false, b = shift2, sel = b[2], out = mul2);
    Add16(a = add1, b = mul2, out = add2);
    ShiftLeft(in = shift2, out = shift3);

    Mux16(a[0..14] = false, b = shift3, sel = b[3], out = mul3);
    Add16(a = add2, b = mul3, out = add3);
    ShiftLeft(in = shift3, out = shift4);

    Mux16(a[0..14] = false, b = shift4, sel = b[4], out = mul4);
    Add16(a = add3, b = mul4, out = add4);
    ShiftLeft(in = shift4, out = shift5);

    Mux16(a[0..14] = false, b = shift5, sel = b[5], out = mul5);
    Add16(a = add4, b = mul5, out = add5);
    ShiftLeft(in = shift5, out = shift6);

    Mux16(a[0..14] = false, b = shift6, sel = b[6], out = mul6);
    Add16(a = add5, b = mul6, out = add6);
    ShiftLeft(in = shift6, out = shift7);

    Mux16(a[0..14] = false, b = shift7, sel = b[7], out = mul7);
    Add16(a = add6, b = mul7, out = add7);
    ShiftLeft(in = shift7, out = shift8);

    Mux16(a[0..14] = false, b = shift8, sel = b[8], out = mul8);
    Add16(a = add7, b = mul8, out = add8);
    ShiftLeft(in = shift8, out = shift9);

    Mux16(a[0..14] = false, b = shift9, sel = b[9], out = mul9);
    Add16(a = add8, b = mul9, out = add9);
    ShiftLeft(in = shift9, out = shift10);

    Mux16(a[0..14] = false, b = shift10, sel = b[10], out = mul10);
    Add16(a = add9, b = mul10, out = add10);
    ShiftLeft(in = shift10, out = shift11);

    Mux16(a[0..14] = false, b = shift11, sel = b[11], out = mul11);
    Add16(a = add10, b = mul11, out = add11);
    ShiftLeft(in = shift11, out = shift12);

    Mux16(a[0..14] = false, b = shift12, sel = b[12], out = mul12);
    Add16(a = add11, b = mul12, out = add12);
    ShiftLeft(in = shift12, out = shift13);

    Mux16(a[0..14] = false, b = shift13, sel = b[13], out = mul13);
    Add16(a = add12, b = mul13, out = add13);
    ShiftLeft(in = shift13, out = shift14);

    Mux16(a[0..14] = false, b = shift14, sel = b[14], out = mul14);
    Add16(a = add13, b = mul14, out = add14);
    ShiftLeft(in = shift14, out = shift15);

    Mux16(a[0..14] = false, b = shift15, sel = b[15], out = mul15);








}
```

8 ShiftLeft.hdl

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

     PARTS:
     Add16(a = in, b = in, out[0..14] = out[0..14], out[15] = ignoredBit);
     // the sign bit
     And(a = in[15], b = true, out = out[15]);
}
```

9 ShiftRight.hdl

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

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