Contents

1 README 2

2 a/Bit.hdl 3

3 a/PC.hdl 4

4 a/RAM64.hdl 5

5 a/RAM8.hdl 6

6 a/Register.hdl 7

7 b/RAM16K.hdl 8

8 b/RAM4K.hdl 9

9 b/RAM512.hdl 10

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 3 - Sequential Logic
                          ----------------------------


Submitted Files
---------------
README        -    This file.
a/Bit.hdl     -    1-bit register.
a/Register.hdl    -    16-bit register.
a/RAM8.hdl        -    Memory of 8 16-bit registers.
a/RAM64.hdl        -    Memory of 64 registers.
b/RAM512.hdl    -    Memory of 512 registers.
b/RAM4K.hdl        -    Memory of 4096 registers.
b/RAM16K.hdl    -    Memory of ~160000 registers.
a/PC.hdl        -    The program counter chip.

Remarks
-------
* The logic in the PC chip was to store the data across time using a register and manipulate
  it in the correct manner using a multiplexer.
* All other chips had simple implementations that were drawn directly from the lectures
  and simple logic.
```

2 a/Bit.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/03/a/Bit.hdl

/**
 * 1-bit register:
 * If load[t] == 1 then out[t+1] = in[t]
 *                 else out does not change (out[t+1] = out[t])
 */

CHIP Bit {
    IN in, load;
    OUT out;

    PARTS:
        // As shown in the lectures...
        Mux(a=ffout, b=in, sel=load, out=ffin);
        DFF(in=ffin, out=out, out=ffout);
}
```

3 a/PC.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/03/a/PC.hdl

/**
 * 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]
 */

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

    PARTS:
        // Increment the output
        Inc16(in=regout, out=incout);

        // According to control bits, decide what will be the register input
        Mux4Way16(a=regout, b=incout, c=in, d=in, sel[0]=inc, sel[1]=load, out=regin1);
        Mux16(a=regin1, b=false, sel=reset, out=regin);

        // If one of the control bits is on, the register should be loaded
        Or8Way(in[0]=inc, in[1]=load, in[2]=reset, in[3..7]=false, out=regload);

        // This is where the magic happens
        Register(in=regin, load=regload, out=regout, out=out);
}
```

4 a/RAM64.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/03/a/RAM64.hdl

/**
 * Memory of 64 registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM64 {
    IN in[16], load, address[6];
    OUT out[16];

    PARTS:
        // Same logic as RAM8

        // According to the address MSBs, decide which RAM to load
        DMux8Way(in=load, sel=address[3..5],
                 a=load0, b=load1, c=load2, d=load3,
                 e=load4, f=load5, g=load6, h=load7);

        // According to the address' LSBs, decide which Register inside the RAM to handle
        RAM8(in=in, load=load0, address=address[0..2], out=out0);
        RAM8(in=in, load=load1, address=address[0..2], out=out1);
        RAM8(in=in, load=load2, address=address[0..2], out=out2);
        RAM8(in=in, load=load3, address=address[0..2], out=out3);
        RAM8(in=in, load=load4, address=address[0..2], out=out4);
        RAM8(in=in, load=load5, address=address[0..2], out=out5);
        RAM8(in=in, load=load6, address=address[0..2], out=out6);
        RAM8(in=in, load=load7, address=address[0..2], out=out7);

        // Again, according to MSBs, decide what data to output
        Mux8Way16(a=out0, b=out1, c=out2, d=out3,
                  e=out4, f=out5, g=out6, h=out7,
                  sel=address[3..5], out=out);
}
```

5 a/RAM8.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/03/a/RAM8.hdl

/**
 * Memory of 8 registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM8 {
    IN in[16], load, address[3];
    OUT out[16];

    PARTS:
        // Decide which Register to handle, according to the address input
        DMux8Way(in=load, sel=address,
                 a=load0, b=load1, c=load2, d=load3,
                 e=load4, f=load5, g=load6, h=load7);

        // Feed the registers
        Register(in=in, load=load0, out=out0);
        Register(in=in, load=load1, out=out1);
        Register(in=in, load=load2, out=out2);
        Register(in=in, load=load3, out=out3);
        Register(in=in, load=load4, out=out4);
        Register(in=in, load=load5, out=out5);
        Register(in=in, load=load6, out=out6);
        Register(in=in, load=load7, out=out7);

        // Decide which register's data should be the output
        Mux8Way16(a=out0, b=out1, c=out2, d=out3,
                  e=out4, f=out5, g=out6, h=out7,
                  sel=address, out=out);
}
```

6 a/Register.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/03/a/Register.hdl

/**
 * 16-bit register:
 * If load[t] == 1 then out[t+1] = in[t]
 * else out does not change
 */

CHIP Register {
    IN in[16], load;
    OUT out[16];

    PARTS:
        // Just a chain of bits
        Bit(in=in[0], load=load, out=out[0]);
        Bit(in=in[1], load=load, out=out[1]);
        Bit(in=in[2], load=load, out=out[2]);
        Bit(in=in[3], load=load, out=out[3]);
        Bit(in=in[4], load=load, out=out[4]);
        Bit(in=in[5], load=load, out=out[5]);
        Bit(in=in[6], load=load, out=out[6]);
        Bit(in=in[7], load=load, out=out[7]);
        Bit(in=in[8], load=load, out=out[8]);
        Bit(in=in[9], load=load, out=out[9]);
        Bit(in=in[10], load=load, out=out[10]);
        Bit(in=in[11], load=load, out=out[11]);
        Bit(in=in[12], load=load, out=out[12]);
        Bit(in=in[13], load=load, out=out[13]);
        Bit(in=in[14], load=load, out=out[14]);
        Bit(in=in[15], load=load, out=out[15]);
}
```

7 b/RAM16K.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/03/b/RAM16K.hdl

/**
 * Memory of 16K registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM16K {
    IN in[16], load, address[14];
    OUT out[16];

    PARTS:
        // Same as previous chips, only a bit smaller (actually, A LOT of bits smaller, but that's abstracted)
        DMux4Way(in=load, sel=address[12..13],
                 a=load0, b=load1, c=load2, d=load3);

        RAM4K(in=in, load=load0, address=address[0..11], out=out0);
        RAM4K(in=in, load=load1, address=address[0..11], out=out1);
        RAM4K(in=in, load=load2, address=address[0..11], out=out2);
        RAM4K(in=in, load=load3, address=address[0..11], out=out3);

        Mux4Way16(a=out0, b=out1, c=out2, d=out3, sel=address[12..13], out=out);
}
```

8 b/RAM4K.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/03/b/RAM4K.hdl

/**
 * Memory of 4K registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM4K {
    IN in[16], load, address[12];
    OUT out[16];

    PARTS:
        // Again, same as RAM512
        DMux8Way(in=load, sel=address[9..11],
                 a=load0, b=load1, c=load2, d=load3,
                 e=load4, f=load5, g=load6, h=load7);

        RAM512(in=in, load=load0, address=address[0..8], out=out0);
        RAM512(in=in, load=load1, address=address[0..8], out=out1);
        RAM512(in=in, load=load2, address=address[0..8], out=out2);
        RAM512(in=in, load=load3, address=address[0..8], out=out3);
        RAM512(in=in, load=load4, address=address[0..8], out=out4);
        RAM512(in=in, load=load5, address=address[0..8], out=out5);
        RAM512(in=in, load=load6, address=address[0..8], out=out6);
        RAM512(in=in, load=load7, address=address[0..8], out=out7);

        Mux8Way16(a=out0, b=out1, c=out2, d=out3,
                  e=out4, f=out5, g=out6, h=out7,
                  sel=address[9..11], out=out);
}
```

9 b/RAM512.hdl

```
// This file is part of the materials accompanying the book
// "The Elements of Computing Systems" by Nisan and Schocken,
// MIT Press. Book site: www.idc.ac.il/tecs
// File name: projects/03/b/RAM512.hdl

/**
 * Memory of 512 registers, each 16 bit-wide. Out holds the value
 * stored at the memory location specified by address. If load==1, then
 * the in value is loaded into the memory location specified by address
 * (the loaded value will be emitted to out from the next time step onward).
 */

CHIP RAM512 {
    IN in[16], load, address[9];
    OUT out[16];

    PARTS:
        // Exactly the same logic as RAM64
        DMux8Way(in=load, sel=address[6..8],
                 a=load0, b=load1, c=load2, d=load3,
                 e=load4, f=load5, g=load6, h=load7);

        RAM64(in=in, load=load0, address=address[0..5], out=out0);
        RAM64(in=in, load=load1, address=address[0..5], out=out1);
        RAM64(in=in, load=load2, address=address[0..5], out=out2);
        RAM64(in=in, load=load3, address=address[0..5], out=out3);
        RAM64(in=in, load=load4, address=address[0..5], out=out4);
        RAM64(in=in, load=load5, address=address[0..5], out=out5);
        RAM64(in=in, load=load6, address=address[0..5], out=out6);
        RAM64(in=in, load=load7, address=address[0..5], out=out7);

        Mux8Way16(a=out0, b=out1, c=out2, d=out3,
                  e=out4, f=out5, g=out6, h=out7,
                  sel=address[6..8], out=out);
}
```