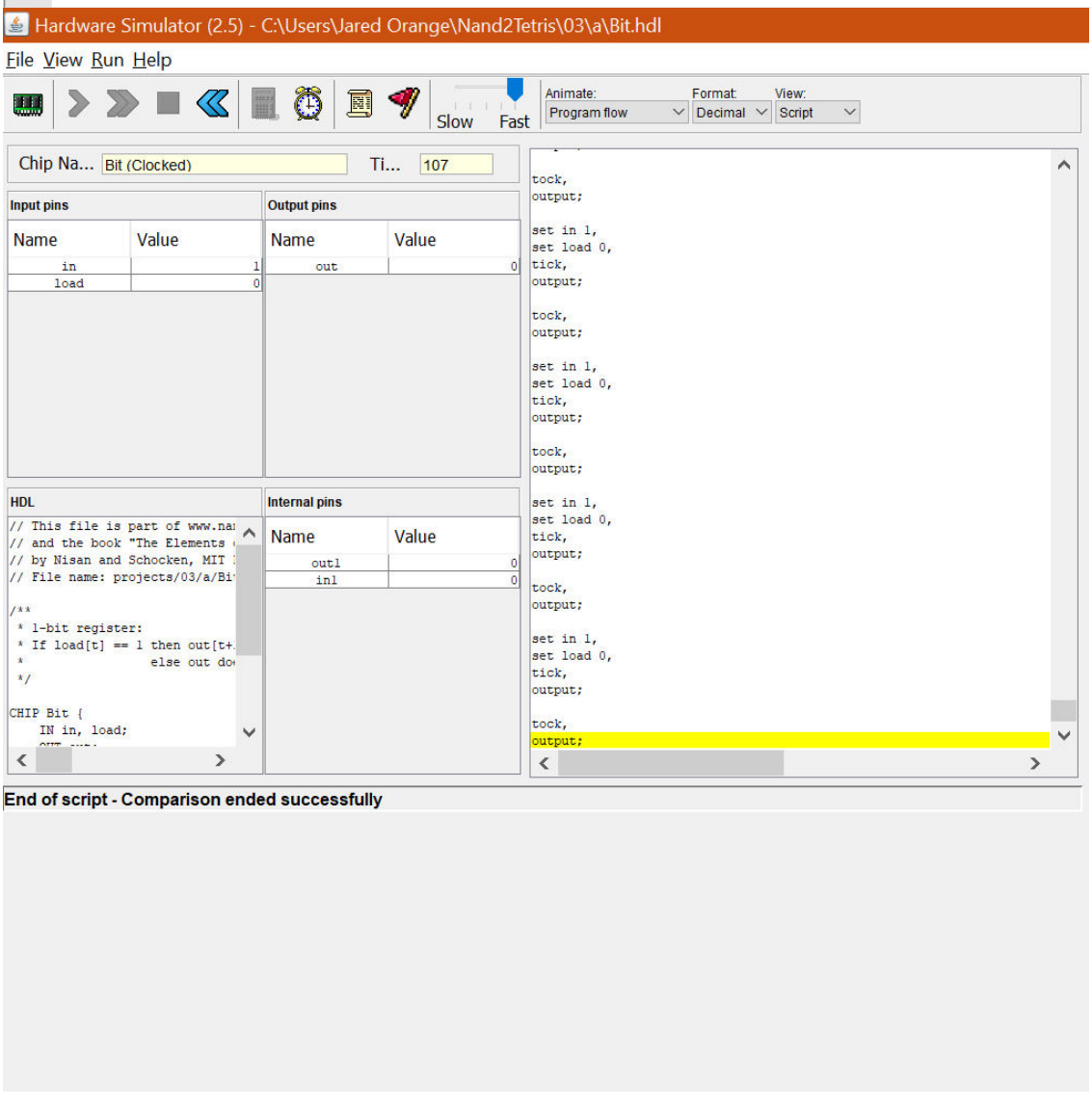
Member: Jared Orange, Marcos Buzn, Wencong Huang

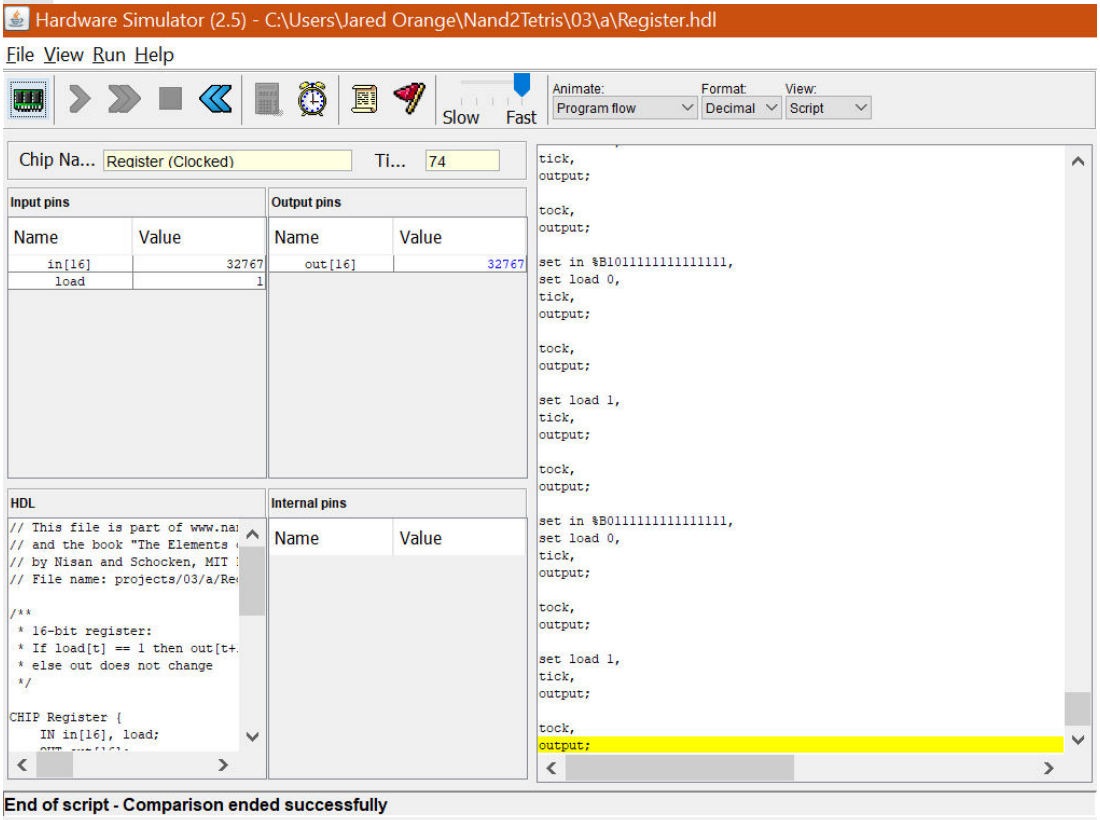
Bit

```
// 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

// File coded by Jared Orange

/**
 * 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:
    // A bit is designated to store a bit of data.
    // It stores this bit until another bit is stored.
    Mux(a=out1, b=in, sel=load, out=in1);
    DFF(in=in1, out=out1, out=out);
}
```



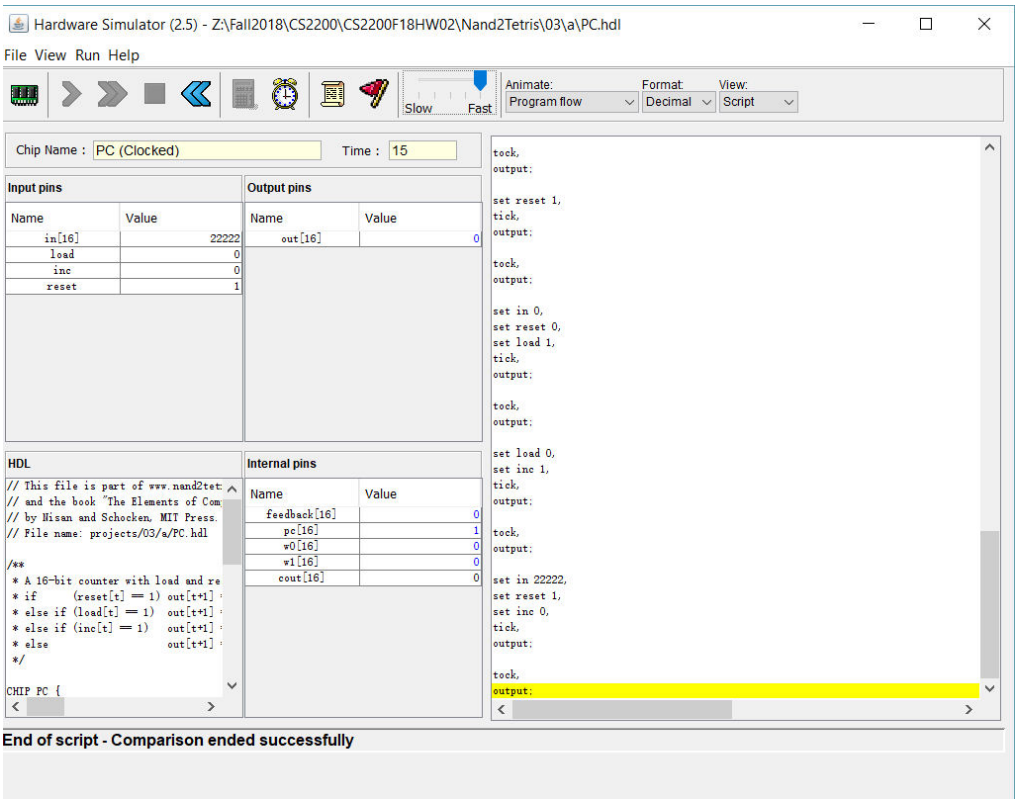
Register

```
 * If load[t] == 1 then out[t+1] = in[t]
 * else out does not change
 */

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

    // DOCUMENTATION //

    // A Register is a storage device used to store values over time. A multi bit register
    // can be constructed from an array of w bit chips (In this case, we are building a 16 bit register
    // so we need 16 1 bit chips). The input pin carries the data from the numeric input and the load allows
    // the array cell to be written in. The output pin carries out the current state of the array cell.
    // The read and write is very similar to the Bit.hdl

    PARTS:
    // Put your code here:
    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]);
}
```



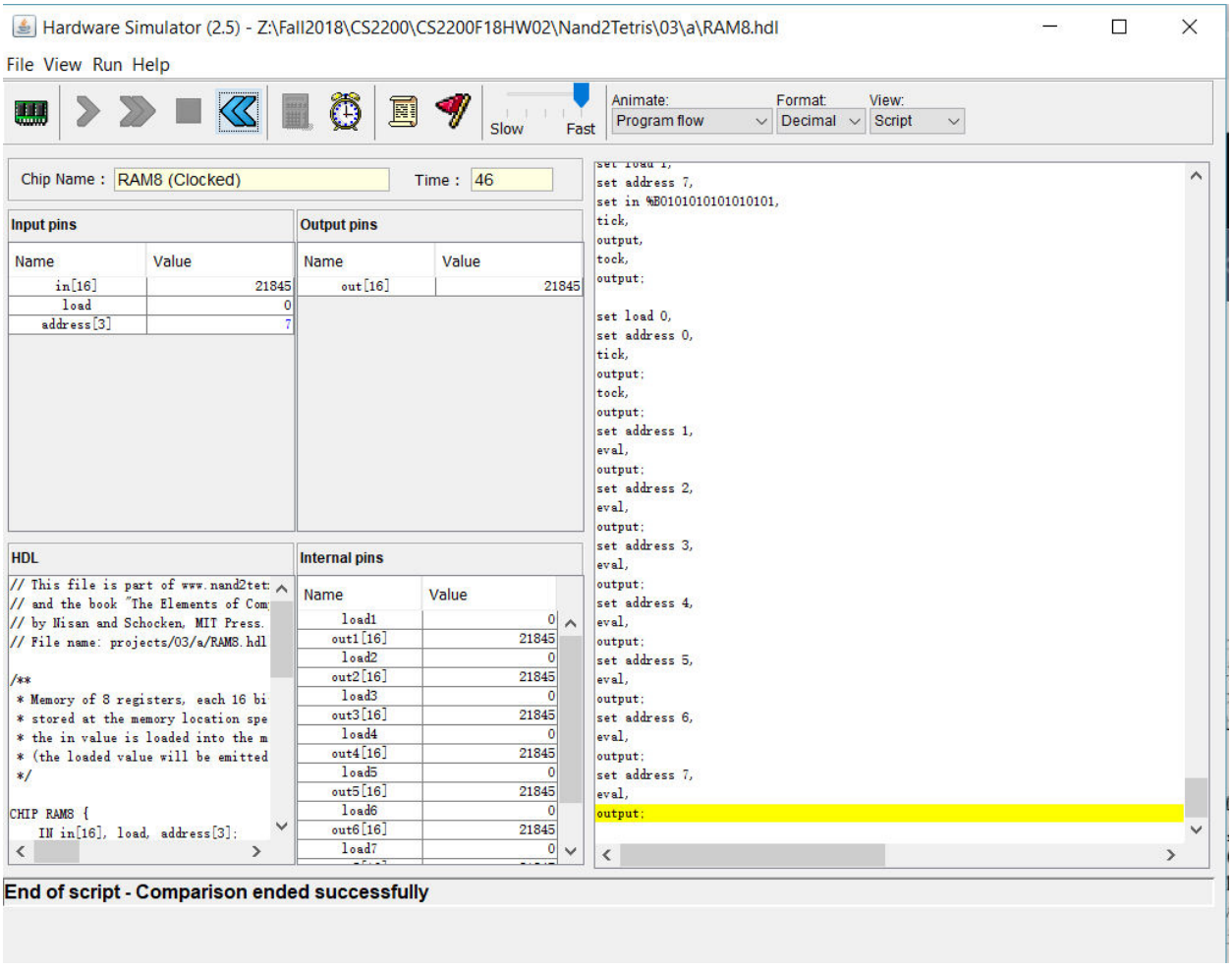
PC

```
/**
 * 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:
    // Put your code here:

    // increment the output of the register
    Inc16(in = feedback, out = pc);

    // "Sequential chips always consist of a layer of DFFs sandwiched
    // between optional combinational logic layers"
    // The next 3 lines are a combinational logic layer to figure
    // out what gets fed to the register. Either the program counter,
    // the incremented pc, the input, or zeros on a reset

    Mux16(a = feedback, b = pc, sel = inc, out = w0);
    Mux16(a = w0, b = in, sel = load, out = w1);
    Mux16(a = w1, b = false, sel = reset, out = cout);

    // the output from the register also needs to get fed back through
    // the combinational logic to get processed for the next clock cycle.
    Register(in = cout, load = true, out = out, out = feedback);
}
```



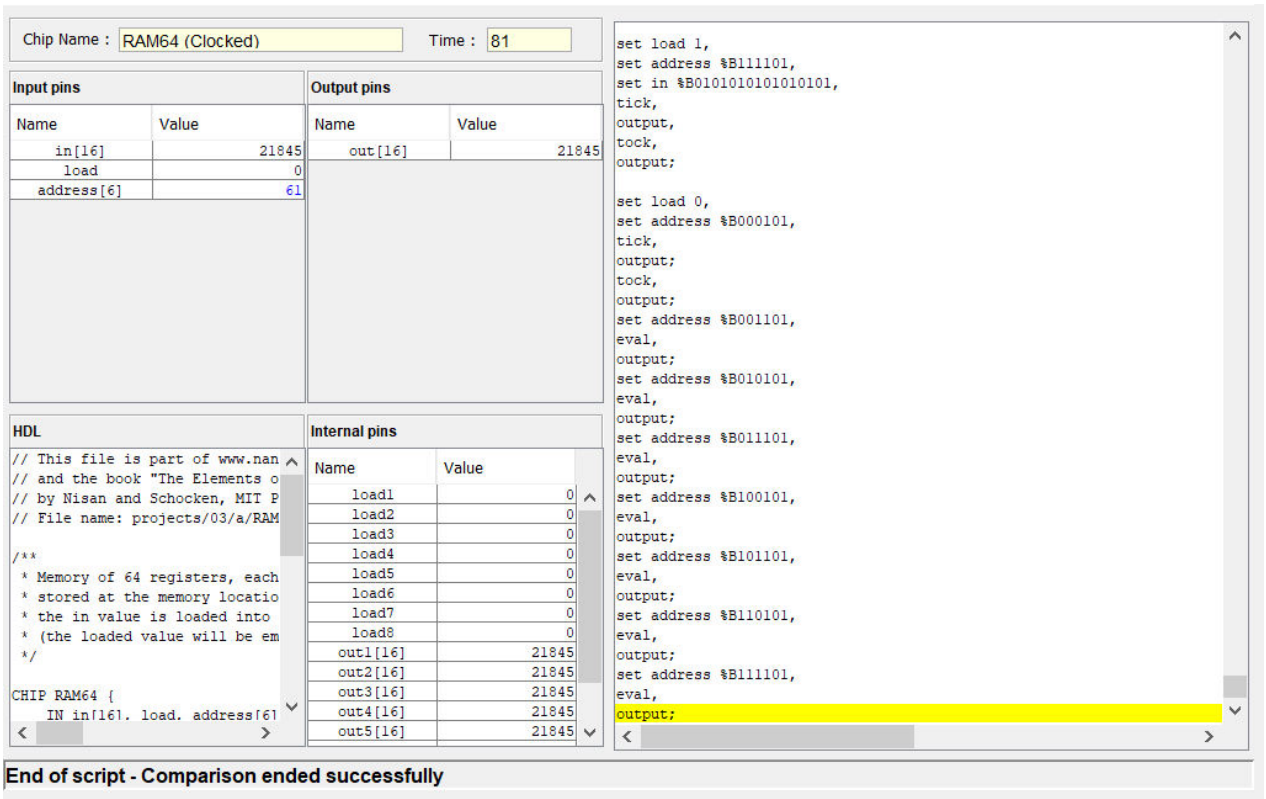
RAM8

```
/**
 * 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:
    // Put your code here:

    // RAM8 will consist of 8 16-bit registers.
    // each register's input is directly connected to the RAM's input
    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);
    Register(in=in, load=load8, out=out8);

    // which register is actually being written to (if any) is given by
    // using a dmux8 to send load to the appropriate register
    DMux8Way(in=load, a=load1, b=load2, c=load3, d=load4, e=load5, f=load6, g=load7, h=load8, sel=address);

    // The final output will be given by
    // using a Mux8Way16 to select which register to output
    Mux8Way16(out=out, a=out1, b=out2, c=out3, d=out4, e=out5, f=out6, g=out7, h=out8, sel=address);
}
```



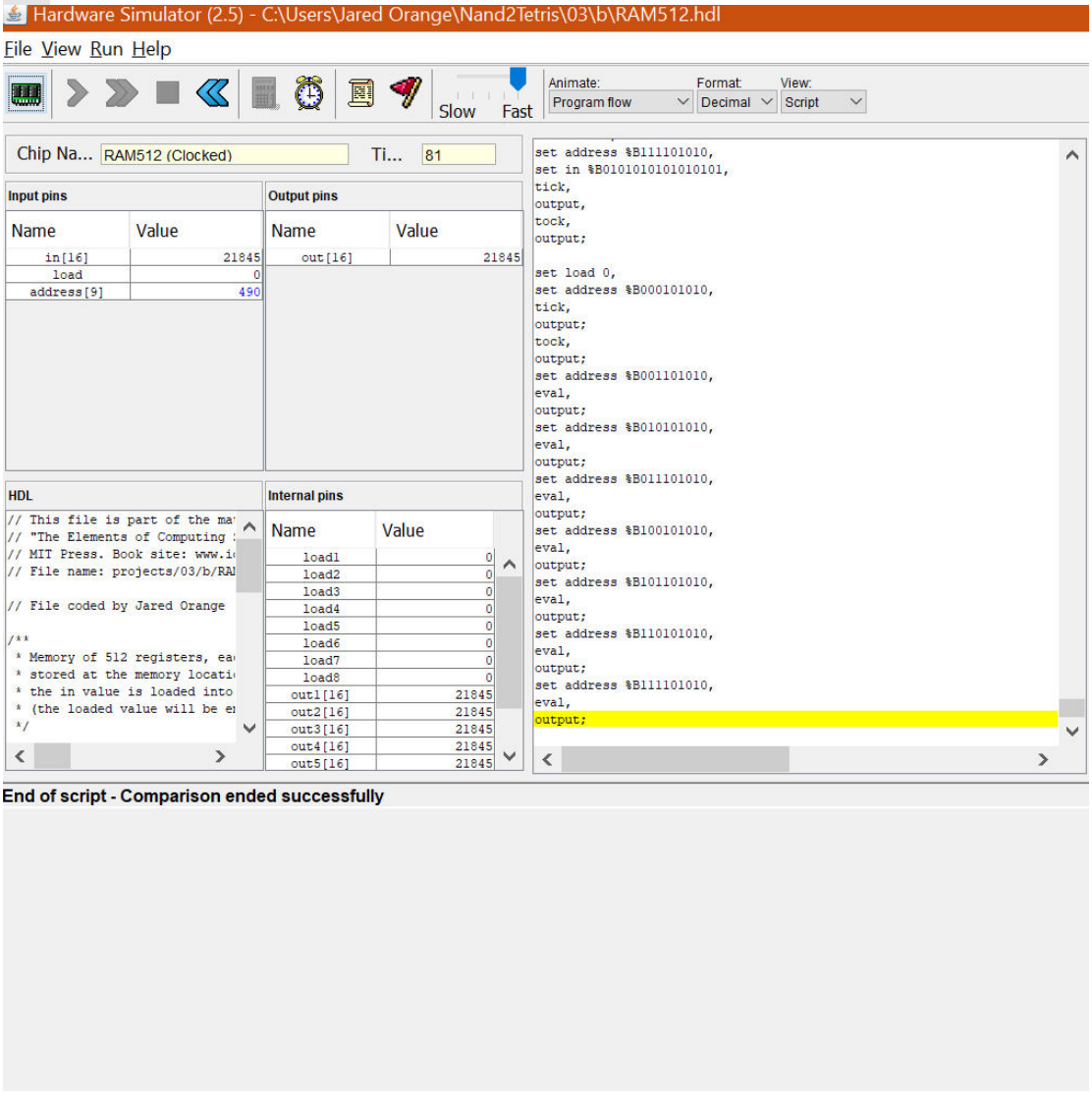
RAM64

```
// 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:
    // RAM64 is an array of 8 RAM8 chips.
    // RAM64 requires the use of 6 registers (2^6)
    // Each RAM must be assigned a unique address (some integer between 0 and n-1) in order to be accessed
    // Each input is loaded corresponding to its data value and the RAM's output will start emitting it
    // In order to select the register specified by the address, we need a DMux8Way and Mux8Way16
    // The DMux8Way and Mux8Way16 would be assigned the remaining 3 address.
    // The DMux8Way would take in the 8 inputs simultaneously and the Mux8Way16 would be fed out the
    // combinational logic circuit.

    DMux8Way(in=load, sel = address[3..5], a=load1, b=load2, c=load3, d=load4, e=load5, f=load6, g=load7, h=load8);
    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);
    RAM8(in=in, load=load8, address=address[0..2], out=out8);
    Mux8Way16(a=out1, b=out2, c=out3, d=out4, e=out5, f=out6, g=out7, h=out8, sel=address[3..5], out=out);
}
```

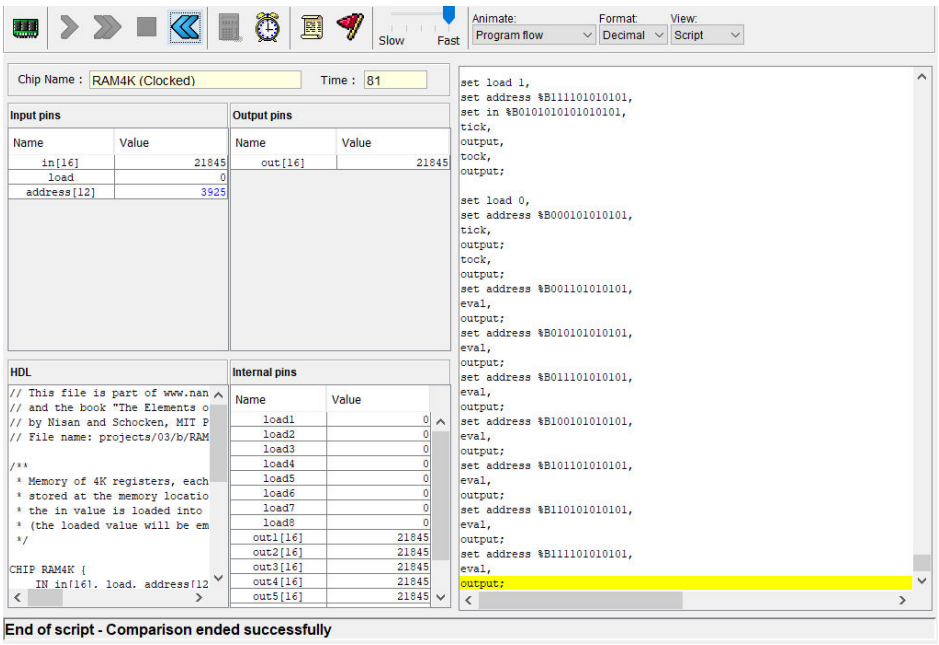


RAM512

```
 * 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:
    // Put your code here:

    //     DOCUMENTATION    //

    // A RAM unit can only be built using smaller RAM parts. A RAM512
    // is an array of 8 RAM64 chips (one below in size). RAM512 requires the use of 6 registers (2^6)
    // Each RAM must be assigned a unique address (some integer between 0 and n-1) in order to be accessed
    // Each input is loaded corresponding to its data value and the RAM's output will start emitting it
    // In order to select the register specified by the address, we need a DMux8Way and Mux8Way16
    // The DMux8Way and Mux8Way16 would be assigned the remaining 2 address.
    // The DMux8Way would take in the 8 inputs simultaneously and the Mux8Way16 would be fed out the
    // combinational logic circuit.

    DMux8Way(in=load, sel=address[6..8], a=load1, b=load2, c=load3, d=load4, e=load5, f=load6, g=load7, h=load8);
    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);
    RAM64(in=in, load=load8, address=address[0..5], out=out8);
    Mux8Way16(a=out1, b=out2, c=out3, d=out4, e=out5, f=out6, g=out7, h=out8, sel=address[6..8], out=out);
}
```



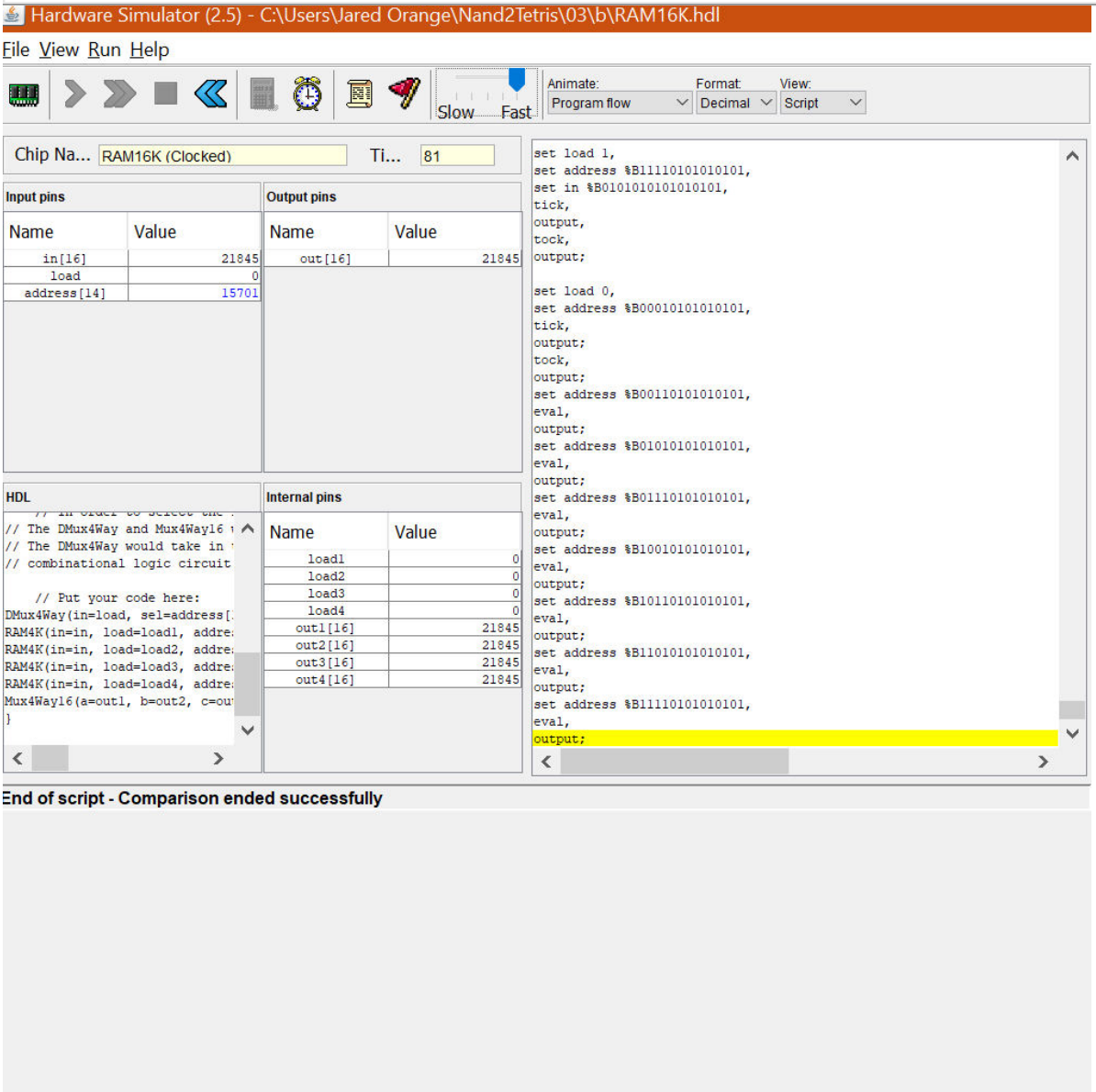
RAM4K




```
// 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:
    // A RAM4K is an array of 8 RAM512 chips.
    // RAM4K requires the use of 12 registers (2^12)
    // Each RAM must be assigned a unique address (some integer between 0 and n-1) in order to be accessed
    // Each input is loaded corresponding to its data value and the RAM's output will start emitting it
    // In order to select the register specified by the address, we need a DMux4Way and Mux4Way16
    // The DMux8Way and Mux8Way16 would be assigned the remaining 3 address.
    // The DMux8Way would take in the 8 inputs simultaneously and the Mux8Way16 would be fed out the
    // combinational logic circuit.
    DMux8Way(in=load, sel=address[9..11], a=load1, b=load2, c=load3, d=load4, e=load5, f=load6, g=load7, h=load8);
    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);
    RAM512(in=in, load=load8, address=address[0..8], out=out8);
    Mux8Way16(a=out1, b=out2, c=out3, d=out4, e=out5, f=out6, g=out7, h=out8, sel=address[9..11], out=out);
}
```

RAM16K

```
// 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

// File coded by Jared Orange

/**
 * 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:
    // A RAM16K is an array of 4 RAM4K chips (one below in size).
    // RAM16K requires the use of 14 registers (2^14)
    // Each RAM must be assigned a unique address (some integer between 0 and n-1) in order to be accessed
    // Each input is loaded corresponding to its data value and the RAM's output will start emitting it
    // In order to select the register specified by the address, we need a DMux4Way and Mux4Way16
    // The DMux4Way and Mux4Way16 would be assigned the remaining 2 address.
    // The DMux4Way would take in the 4 inputs simultaneously and the Mux4Way16 would be fed out the
    // combinational logic circuit.

    // Put your code here:
    DMux4Way(in=load, sel=address[12..13], a=load1, b=load2, c=load3, d=load4);
    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);
    RAM4K(in=in, load=load4, address=address[0..11], out=out4);
    Mux4Way16(a=out1, b=out2, c=out3, d=out4, sel=address[12..13], out=out);
}
```



Are the test adequate or inadequate?

RAMs: The RAM tests are adequate. The sole purpose of RAM is to provide fast read/write access to a storage device. The tst files load in an address and use ticks and tocks to count how fast is reads/writes to the device by displaying in between functions. If anything, we would have added some sort of device to calculate speed in MHz (like typical RAM inside a computer is measured), a test to find the max slots used when read/writing to the device and then a test comparing the runtime versus an actual hard drive.

Bit: Adequate. All that a bit does is store data and execute instructions in bytes. The tst file sets a value, loads it, starts counting using ticks and tocks and outputs the executable time of storing the data.

Register: Inadequate. Register and Bit have the same read/write functions, so testing Bit first is all that is necessary since n Register is made up of n Bits.

PC: Adequate. The Program Counter is working with the reset, load, and inc, (the if-else statement in the comment section) total of 3 bits and that gives us 2^3=8 cases. In the test file, it had checked all the cases/combinations of reset, load, and inc. Therefore, the test is adequate.