```
// 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/05/Memory.hdl

/**
 * The complete address space of the Hack computer's memory,
 * including RAM and memory-mapped I/O.
 * The chip facilitates read and write operations, as follows:
 *     Read:  out(t) = Memory[address(t)](t)
 *     Write: if load(t-1) then Memory[address(t-1)](t) = in(t-1)
 * In words: the chip always outputs the value stored at the memory
 * location specified by address. If load==1, the in value is loaded
 * into the memory location specified by address. This value becomes
 * available through the out output from the next time step onward.
 * Address space rules:
 * Only the upper 16K+8K+1 words of the Memory chip are used.
 * Access to address>0x6000 is invalid. Access to any address in
 * the range 0x4000-0x5FFF results in accessing the screen memory
 * map. Access to address 0x6000 results in accessing the keyboard
 * memory map. The behavior in these addresses is described in the
 * Screen and Keyboard chip specifications given in the book.
 */

CHIP Memory {
    IN in[16], load, address[15];
    OUT out[16];

    PARTS:
    // Put your code here:
    DMux4Way(in=load, sel=address[13..14], a=x1 ,b=x2 ,c=x3 ,d=x4);
    Or(a=x1, b=x2, out=y1);
    RAM16K(in=in, load=y1, address=address[0..13], out=w1);
    Screen(in=in, load=x3, address=address[0..12], out=w2);
    Keyboard(out=w3);
    Mux4Way16(a=w1, b=w1, c=w2, d=w3, sel=address[13..14], out=out);
}
```

```
// 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/05/CPU.hdl

/**
 * The Hack CPU (Central Processing unit), consisting of an ALU,
 * two registers named A and D, and a program counter named PC.
 * The CPU is designed to fetch and execute instructions written in
 * the Hack machine language. In particular, functions as follows:
 * Executes the inputted instruction according to the Hack machine
 * language specification. The D and A in the language specification
 * refer to CPU-resident registers, while M refers to the external
 * memory location addressed by A, i.e. to Memory[A]. The inM input
 * holds the value of this location. If the current instruction needs
 * to write a value to M, the value is placed in outM, the address
 * of the target location is placed in the addressM output, and the
 * writeM control bit is asserted. (When writeM==0, any value may
 * appear in outM). The outM and writeM outputs are combinational:
 * they are affected instantaneously by the execution of the current
 * instruction. The addressM and pc outputs are clocked: although they
 * are affected by the execution of the current instruction, they commit
 * to their new values only in the next time step. If reset==1 then the
 * CPU jumps to address 0 (i.e. pc is set to 0 in next time step) rather
 * than to the address resulting from executing the current instruction.
 */

CHIP CPU {

    IN  inM[16],         // M value input  (M = contents of RAM[A])
        instruction[16], // Instruction for execution
        reset;           // Signals whether to re-start the current
                         // program (reset==1) or continue executing
                         // the current program (reset==0).

    OUT outM[16],        // M value output
        writeM,          // Write to M?
        addressM[15],    // Address in data memory (of M)
        pc[15];          // address of next instruction

    PARTS:
    // Put your code here:
    //decode and Mux1
    Not(in=instruction[15], out=notinstruction);
    Mux16(a=ALUoutput, b=instruction, sel=notinstruction, out=w1);

    //Aregister
    Or(a=notinstruction, b=instruction[5], out=a1);
    ARegister(in=w1, load=a1,out=aout, out[0..14]=addressM);

    //Mux2
    Mux16(a=aout, b=inM, sel=instruction[12], out=am);

    //ALU
    ALU(x=d ,y=am, zx=instruction[11], nx=instruction[10], zy=instruction[9],
ny=instruction[8], f=instruction[7], no=instruction[6], out=ALUoutput, out=outM,
zr=zr, ng=ng);

    //Dregister
    And(a=instruction[15], b=instruction[4], out=d1);
    DRegister(in=ALUoutput, load=d1, out=d);

    //ZR
    And(a=instruction[1], b=instruction[15], out=zr1);
    And(a=zr1, b=zr, out=jmpzr);

    //NG
    And(a=instruction[2], b=instruction[15], out=ng1);
    And(a=ng1, b=ng, out=jmpng);

    Not(in=zr, out=notzr);
    Not(in=ng, out=notng);
    And(a=instruction[0], b=instruction[15], out=cgreater0);
    And(a=notzr, b=notng, out=posnum);
    And(a=cgreater0, b=posnum, out=jmppos);

    //jump
    Or(a=jmpzr, b=jmpng, out=jmpzrng);
    Or(a=jmpzrng, b=jmppos, out=jump);

    //writeM
    And(a=instruction[15],b=instruction[3], out=writeM);

    //pc
    Not(in=jump, out=notjump);
    PC(in=aout, load=jump, inc=notjump, reset=reset, out[0..14]=pc);




}
```

```
// 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/05/Computer.hdl

/**
 * The HACK computer, including CPU, ROM and RAM.
 * When reset is 0, the program stored in the computer's ROM executes.
 * When reset is 1, the execution of the program restarts.
 * Thus, to start a program's execution, reset must be pushed "up" (1)
 * and "down" (0). From this point onward the user is at the mercy of
 * the software. In particular, depending on the program's code, the
 * screen may show some output and the user may be able to interact
 * with the computer via the keyboard.
 */

CHIP Computer {

    IN reset;

    PARTS:
    // Put your code here:
    ROM32K(address=pc , out=instruction);
    CPU(inM=inM, instruction=instruction, reset=reset, outM=outM, writeM=writeM,
addressM=addressM, pc=pc);
    Memory(in=outM, load=writeM, address=addressM,out=inM);
}
```