

```
1 package io;
2
3 import memory.DRegister;
4 import memory.Register;
5
6 import java.io.*;
7 import java.util.Scanner;
8
9 public class HackIO implements IOInterface {
10     private InputStream inputStream;
11     private PrintStream outputStream;
12
13     public HackIO(InputStream in, PrintStream out) {
14         inputStream = in;
15         outputStream = out;
16     }
17
18     @Override
19     public Register readData() {
20         Scanner scanner = new Scanner(inputStream);
21         short input = scanner.nextShort(); // reads from input
22         return new DRegister(input);
23     }
24
25
26     @Override
27     public void writeData(Register data) {
28         outputStream.println(data.getValue());
29     }
30
31 }
32
```

```
1 package io;
2
3 import memory.Register;
4
5 public interface IOInterface {
6     Register readData();
7
8     void writeData(Register data);
9 }
10
```

```
1 package alu;
2
3 import memory.Register;
4
5 public interface ALU {
6
7     Register compute(Register r1, Register r2);
8
9     void setFlagF(boolean flagF);
10
11     boolean isFlagZr();
12
13     boolean isFlagNg();
14 }
15
```

```

1 package alu;
2
3 import memory.DRegister;
4 import memory.Register;
5
6 public class HackALU implements ALU {
7
8     // Input control flag, only one flag implemented for simplicity.
9     private boolean flagF; // function flag
10
11     // Output flags
12     private boolean flagZr; // output zero flag
13     private boolean flagNg; // output negative flag
14
15     public Register compute(Register r1, Register r2) {
16         Register result;
17
18         if (flagF) {
19             System.out.println("ALU performing ADD operation");
20             result = add(r1, r2);
21         } else {
22             System.out.println("ALU performing SUBTRACT operation");
23             result = sub(r1, r2);
24         }
25         if (result.getValue() == 0) {
26             flagZr = true;
27         } else if (result.getValue() < 0) {
28             flagNg = true;
29         }
30         return result;
31     }
32 }
33
34 public Register add(Register r1, Register r2) {
35     short r1Value = r1.getValue();
36     short r2Value = r2.getValue();
37     short sum = (short) (r1Value + r2Value);
38     DRegister result = new DRegister(sum);
39     return result;
40 }
41
42 public Register sub(Register r1, Register r2) {
43     short r1Value = r1.getValue();
44     short r2Value = r2.getValue();
45     short sum = (short) (r1Value - r2Value);
46     DRegister result = new DRegister(sum);
47     return result;
48 }
49
50
51 public boolean isFlagF() {
52     return flagF;
53 }
54
55 @Override
56 public void setFlagF(boolean flagF) {
57     this.flagF = flagF;
58 }
59
60 @Override
61 public boolean isFlagZr() {
62     return flagZr;
63 }
64
65 @Override
66 public boolean isFlagNg() {
67     return flagNg;

```

```
68     }  
69 }  
70
```

```
1 package cpu;
2
3 import alu.ALU;
4 import memory.ARegister;
5 import memory.DRegister;
6
7 import java.util.concurrent.atomic.AtomicInteger;
8
9 public interface CPU {
10     ALU getAlu();
11
12     AtomicInteger getProgramCounter();
13
14     ARegister getaRegister();
15
16     DRegister getdRegister();
17
18     void setaRegister(ARegister aReg);
19
20     void setdRegister(DRegister dReg);
21 }
22
```

```
1 package cpu;
2
3 import alu.HackALU;
4 import alu.ALU;
5 import memory.ARegister;
6 import memory.DRegister;
7
8 import java.util.concurrent.atomic.AtomicInteger;
9
10 public class HackCPU implements CPU {
11     private ALU alu;
12     private AtomicInteger programCounter;
13     private ARegister aRegister;
14     private DRegister dRegister;
15
16     public HackCPU() {
17         this.alu = new HackALU();
18         this.programCounter = new AtomicInteger(0);
19         this.aRegister = new ARegister((short) 0);
20         this.dRegister = new DRegister((short) 0);
21     }
22
23     @Override
24     public ALU getAlu() {
25         return alu;
26     }
27
28     public void setAlu(ALU alu) {
29         this.alu = alu;
30     }
31
32     @Override
33     public AtomicInteger getProgramCounter() {
34         return programCounter;
35     }
36
37     public void setProgramCounter(AtomicInteger programCounter) {
38         this.programCounter = programCounter;
39     }
40
41     @Override
42     public ARegister getaRegister() {
43         return aRegister;
44     }
45
46     public void setaRegister(ARegister aRegister) {
47         this.aRegister = aRegister;
48     }
49
50     @Override
51     public DRegister getdRegister() {
52         return dRegister;
53     }
54
55     public void setdRegister(DRegister dRegister) {
56         this.dRegister = dRegister;
57     }
58 }
59
```

```
1 package gates;
2
3 /**
4  * Or gate: compute = Not(a) Nand Not(b)
5  * compute = 1 if (a == 1 or b == 1)
6  *           0 otherwise
7  */
8
9 public class ORGate extends BinaryGate {
10
11     // Constructor
12     public ORGate(boolean a, boolean b) {
13         super(a, b);
14     }
15
16     // Implementation of Gate interface
17     @Override
18     public boolean out() {
19         return a || b;
20     }
21 }
22
```



```
1 package gates;
2
3 /**
4  * And gate:
5  * compute = 1 if (a == 1 and b == 1)
6  *           0 otherwise
7  *
8  * true represents 1 and false 0
9  */
10 public class ANDGate extends BinaryGate {
11
12     public ANDGate(boolean a, boolean b) {
13         super(a, b);
14     }
15
16     @Override
17     public boolean out() {
18         return a && b;
19     }
20 }
21
```

```
1 package gates;
2
3 public class NOTGate extends UnaryGate {
4
5     // Constructor
6     public NOTGate(boolean a) {
7         super(a);
8     }
9
10
11     @Override
12     public boolean out() {
13         return (!a);
14     }
15 }
16
```

```
1 package gates;
2
3 public abstract class UnaryGate implements GateInterface {
4     protected boolean a;
5
6     protected UnaryGate(boolean a) {
7         this.a = a;
8     }
9
10 }
11
```

```
1 package gates;
2
3 public abstract class BinaryGate implements GateInterface {
4     protected boolean a, b;
5     protected BinaryGate (boolean a, boolean b) {
6         this.a = a;
7         this.b = b;
8     }
9 }
10
```

```
1 package gates;  
2  
3 public interface GateInterface {  
4     boolean out();  
5 }  
6
```

```
1 package memory;
2
3 public class Memory implements MemoryInterface {
4     private Register[] registerCells;
5     private short size;
6
7     // Default Constructor
8     public Memory() {
9         this(DEFAULT_SIZE);
10    }
11
12    // Overloading concept for constructor.
13    public Memory(short size) {
14        this.size = size;
15        this.registerCells = new Register[size];
16        for (int i = 0; i < size; i++) {
17            registerCells[i] = new DRegister((short) 0);
18        }
19    }
20
21    @Override
22    public int getSize() {
23        return this.size;
24    }
25
26    @Override
27    public Register fetch(ARegister aRegister) throws MemoryException {
28        short address = aRegister.getValue();
29        if (address > registerCells.length) {
30            throw new MemoryException("Address index of bounds" + address);
31        }
32        return registerCells[address];
33    }
34
35    @Override
36    public void store(ARegister aRegister, DRegister dRegister) throws MemoryException
37    {
38        short address = aRegister.getValue();
39        try {
40            registerCells[address] = dRegister;
41        } catch (ArrayIndexOutOfBoundsException e) {
42            throw new MemoryException("Address index of bounds");
43        }
44    }
45 }
46
```

```
1 package memory;
2
3 public class Register {
4     private short value;
5
6     protected Register(short value) {
7         this.value = value;
8     }
9
10    public short getValue() {
11        return value;
12    }
13 }
14
```

```
1 package memory;
2
3 public class ARegister extends Register {
4
5     public ARegister(short value) {
6         super(value);
7     }
8 }
9
```



```
1 package memory;
2
3 public class DRegister extends Register {
4
5     public DRegister(short value) {
6         super(value);
7     }
8 }
9
```

```
1 package memory;
2
3 public class MemoryException extends Exception {
4     public MemoryException(String s) {
5         super(s);
6     }
7 }
8
```

```
1 package memory;
2
3 public interface MemoryInterface {
4
5     short DEFAULT_SIZE = 100;
6
7     int getSize();
8
9     Register fetch(ARegister address) throws MemoryException;
10
11     void store(ARegister address, DRegister data) throws MemoryException;
12 }
13
```

```
1 package machine;
2
3 import cpu.CPU;
4 import cpu.HackCPU;
5 import io.HackIO;
6 import io.IOInterface;
7 import memory.Memory;
8
9 public class HackMachine {
10
11     private Memory memory;
12     private CPU cpu;
13     private IOInterface io;
14
15     public HackMachine() {
16         this.memory = new Memory((short)4000);
17         this.cpu = new HackCPU();
18         this.io = new HackIO(System.in, System.out);
19     }
20
21     public Memory getMemory() {
22         return memory;
23     }
24
25     public void setMemory(Memory memory) {
26         this.memory = memory;
27     }
28
29     public CPU getCpu() {
30         return cpu;
31     }
32
33     public void setCpu(CPU cpu) {
34         this.cpu = cpu;
35     }
36
37     public IOInterface getIo() {
38         return io;
39     }
40
41     public void setIo(IOInterface io) {
42         this.io = io;
43     }
44 }
45
```

```

1 package machine;
2
3 import alu.ALU;
4 import cpu.CPU;
5 import io.IOInterface;
6 import memory.*;
7
8 import java.util.Scanner;
9 import java.util.concurrent.atomic.AtomicInteger;
10
11 public class TestHackMachine {
12
13     public static void main(String[] args) {
14
15         HackMachine hackMachine = new HackMachine();
16         CPU cpu = hackMachine.getCpu();
17         ALU alu = cpu.getAlu();
18         Memory mem = hackMachine.getMemory();
19         IOInterface io = hackMachine.getIo();
20
21         /* ***** Testing ALU ***** */
22         System.out.println("\n*****Testing HackALU *****");
23         Scanner sc = new Scanner(System.in);
24         System.out.println("Enter input x for ALU operation :");
25         short x = sc.nextShort();
26         System.out.println("Enter input y for ALU operation :");
27         short y = sc.nextShort();
28         DRegister r1 = new DRegister(x);
29         DRegister r2 = new DRegister(y);
30         System.out.println("Enter control flag for function 1 for additon, 0 for
subtraction :");
31         short f = sc.nextShort();
32         if (f == 1)
33             alu.setFlagF(true);
34         else
35             alu.setFlagF(false);
36         System.out.println("ALU output " + alu.compute(r1, r2).getValue());
37         System.out.println("ALU output flag Zr; whether output is zero: " + alu.isFlagZr
());
38         System.out.println("ALU output flag Ng; whether output is negative: " + alu.
isFlagNg());
39
40
41         /* ***** Testing Memory ***** */
42         System.out.println("\n*****Testing Hack memory *****");
43         System.out.println("Memory size is " + mem.getSize());
44         System.out.println("Enter memory location to store:");
45         short loc = sc.nextShort();
46         System.out.println("Enter data to store:");
47         short value = sc.nextShort();
48         ARegister address = new ARegister(loc);
49         DRegister data = new DRegister(value);
50
51         try {
52             mem.store(address, data);
53         } catch (MemoryException e) {
54             System.out.println("Exception in store method - " + e.getMessage());
55         }
56         System.out.println("Store Success in the memory location");
57
58         System.out.println("Enter memory location to fetch:");
59         loc = sc.nextShort();
60         address = new ARegister(loc);
61         try {
62             Register dataFetched = mem.fetch(address);
63             System.out.println("Fetch Success from memory location - value =" +
dataFetched.getValue());

```

```

64     } catch (MemoryException e) {
65         System.out.println("Exception in store method - " + e.getMessage());
66     }
67
68     /* ***** Testing IO ***** */
69     System.out.println("\n***** Testing Hack IO Unit ***** ");
70
71     System.out.println("Enter number on InputStream ");
72     Register input = io.readData();
73     System.out.println("Read from input = " + input.getValue());
74     System.out.println("Writing to output is ");
75     io.writeData(input);
76
77
78     /* *****Testing CPU***** */
79     System.out.println("\n*****Testing CPU*****");
80     AtomicInteger programCounter = cpu.getProgramCounter();
81     int nextInstruction = programCounter.getAndIncrement();
82     // ROM[0] has A instruction like @0
83     ARegister aReg = new ARegister((short) 0);
84     System.out.println("Setting instruction @0 into ARegister ");
85     cpu.setARegister(aReg);
86     // D=M means D = RAM[0]; Lets say RAM[0] has value 0;
87     DRegister dReg = new DRegister((short) 0);
88     System.out.println("Setting instruction D=M into DRegister ");
89     cpu.setDRegister(dReg);
90     System.out.println("Executing instruction @ ROM[" + nextInstruction + "]");
91     nextInstruction = cpu.getProgramCounter().getAndIncrement();
92     System.out.println("Executing instruction @ ROM[" + nextInstruction + "]");
93
94 }
95 }
96

```