```java
package cacheDemoProject;

import java.util.*;

public class CacheDemo {

        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.println("Options for cache mapping (All options will use LRU policy)");
                System.out.println("(1) direct mapped (2) 2-way set associative (3) 4-way set associative (4) full associative");
                int associative = input.nextInt();

                /*Setting up an address list to use in the program.
                 *These values represent the memory addresses.
                 *Ex. block address 8 have a bit 32-bit address "0...01000"
                 */
                Map<Integer, String> addressList = new HashMap<>();
                addressList.put(0, "0000");
                addressList.put(1, "0001");
                addressList.put(2, "0010");
                addressList.put(3, "0011");
                addressList.put(4, "0100");
                addressList.put(5, "0101");
                addressList.put(6, "0110");
                addressList.put(7, "0111");
                addressList.put(8, "1000");
                addressList.put(9, "1001");
                addressList.put(10, "1010");
                addressList.put(11, "1011");
                addressList.put(12, "1100");
                addressList.put(13, "1101");
                addressList.put(14, "1110");
                addressList.put(15, "1111");

                double count = 0;
                double countMiss = 0;
                double missRate = 0;

                Deque cache0 = new LinkedList();
                Deque cache1 = new LinkedList();
                Deque cache2 = new LinkedList();
                Deque cache3 = new LinkedList();
                //**Direct mapping**
                if(associative == 1) {
                        System.out.println("direct mapping selected");
                        int n = 0;
                        while(n >= 0 && n <= 15) {
                                //Get input n, check for a match, then fill in cache slot with n.
                                n = input.nextInt();
                                //Stop if input is not within 0-15. Continue otherwise.
                                if(n < 0 || n > 15)
                                        break;
                                //check for match. If match found, it's a hit. Otherwise miss.
                                else if(addressList.get(n) == cache0.peekFirst()
                                                || addressList.get(n) == cache1.peekFirst()
                                                || addressList.get(n) == cache2.peekFirst()
                                                || addressList.get(n) == cache3.peekFirst()) {
                                        System.out.println("Hit");
                                }
                                else {
                                        System.out.println("Miss");
                                        countMiss += 1;
                                }
                                count += 1;

                                /*input n will go into one of the 4 cache slots. If deque size > 1 (cache slot if full), overwrite with n.
                                 *numbers 0, 4, 8, 12 have index 00 so they go into cache0
                                 */
                                if(n % 4 == 0) {
                                        if(cache0.size() > 1) {
                                                cache0.addFirst(addressList.get(n));
                                                cache0.removeLast();
                                                System.out.println("recently accessed block address in cache0 is Mem[" + n + "]");
                                        }
                                        else {
                                                cache0.addFirst(addressList.get(n));
                                                System.out.println("recently accessed block address in cache0 is Mem[" + n + "]");
                                        }
                                }
                                //numbers 1, 5, 9, 13 have index 01 so they go into cache1
                                if(n % 4 == 1) {
                                        if(cache1.size() > 1) {
                                                cache1.addFirst(addressList.get(n));
                                                cache1.removeLast();
                                                System.out.println("recently accessed block address in cache1 is Mem[" + n + "]");
                                        }
                                        else {
                                                cache1.addFirst(addressList.get(n));
                                                System.out.println("recently accessed block address in cache1 is Mem[" + n + "]");
                                        }
                                }
                                //numbers 2, 6, 10, 14 have index 10 so they go into cache2
                                if(n % 4 == 2) {
                                        if(cache2.size() > 1) {
                                                cache2.addFirst(addressList.get(n));
                                                cache2.removeLast();
                                                System.out.println("recently accessed block address in cache2 is Mem[" + n + "]");
                                        }
                                        else {
                                                cache2.addFirst(addressList.get(n));
                                                System.out.println("recently accessed block address in cache2 is Mem[" + n + "]");
                                        }
                                }
                                //numbers 3, 7, 11, 15 have index 11 so they go into cache3
                                if(n % 4 == 3) {
                                        if(cache3.size() > 1) {
                                                cache3.addFirst(addressList.get(n));
                                                cache3.removeLast();
                                                System.out.println("recently accessed block address in cache3 is Mem[" + n + "]");
                                        }
                                        else {
                                                cache3.addFirst(addressList.get(n));
                                                System.out.println("recently accessed block address in cache3 is Mem[" + n + "]");
                                        }
                                }
                        }
                }

                //**2 way set associative mapping**
                Deque twoWaySet0 = new LinkedList();
                Deque twoWaySet1 = new LinkedList();
                Deque twoWaySet2 = new LinkedList();
                Deque twoWaySet3 = new LinkedList();
                if(associative == 2) {
                        System.out.println("2-way associative selected");
                        Boolean hit = false;
                        int n = 0;
                        while(n >= 0 && n <= 15) {
                                n = input.nextInt();
                                //Stop if input n is not within numbers 0-15. Continue otherwise.
                                if(n < 0 || n > 15)
                                        break;
                                //check for a match between n and a cached address. If match found, hit. Otherwise, miss.
                                else if(addressList.get(n) == twoWaySet0.peekFirst() || addressList.get(n) == twoWaySet0.peekLast()) {
                                        System.out.println("Hit");
                                        hit = true;
                                }
                                else if(addressList.get(n) == twoWaySet1.peekFirst() || addressList.get(n) == twoWaySet1.peekLast()) {
                                        System.out.println("Hit");
                                        hit = true;
                                }
                                else if(addressList.get(n) == twoWaySet2.peekFirst() || addressList.get(n) == twoWaySet2.peekLast()) {
                                        System.out.println("Hit");
                                        hit = true;
                                }
                                else if(addressList.get(n) == twoWaySet3.peekFirst() || addressList.get(n) == twoWaySet3.peekLast()) {
                                        System.out.println("Hit");
                                        hit = true;
                                }
                                else {
                                        System.out.println("Miss");
                                        hit = false;
                                        countMiss += 1;
                                }
                                count += 1;

                                /* Which set an address number is stored depends on what the remainder is after dividing the number by 4.
                                 * Ex. 4 % 4 = 0, So Mem[4] is stored in set 0 (twoWaySet0)
                                 */
                                if(hit) {//hit branch
                                        if(n % 4 == 0) {
                                                if(twoWaySet0.size() <= 1)
                                                        System.out.println("Recently accessed block addresses in set 0: Mem["
                                                                        + Integer.parseInt((String)twoWaySet0.peekFirst(), 2) + "]");
                                                else {
                                                        if(n == Integer.parseInt((String)twoWaySet0.peekFirst(), 2)){//accessed address
matches most recent value.
                                                                //Do nothing
                                                        }
                                                        else if(n == Integer.parseInt((String)twoWaySet0.peekLast(), 2)){//accessed address
matches the LRU value.
                                                                twoWaySet0.remove(addressList.get(n));
                                                                twoWaySet0.addFirst(addressList.get(n));
                                                        }
                                                        //Print out content of set 0
                                                        System.out.println("Recently accessed block addresses in set 0: Mem["
                                                                        + Integer.parseInt((String)twoWaySet0.peekFirst(), 2) + "], Mem["
                                                                        + Integer.parseInt((String)twoWaySet0.peekLast(), 2) + "]");
                                                }
                                        }
                                        else if(n % 4 == 1){
                                                if(twoWaySet1.size() <= 1)
                                                        System.out.println("Recently accessed block addresses in set 1: Mem["
                                                                        + Integer.parseInt((String)twoWaySet1.peekFirst(), 2) + "]");
                                                else {
                                                        if(n == Integer.parseInt((String)twoWaySet1.peekFirst(), 2)){//accessed address
matches most recent value.

                                                                //Do nothing
                                                        }
                                                        else if(n == Integer.parseInt((String)twoWaySet1.peekLast(), 2)){//accessed address
matches the LRU value.

                                                                twoWaySet1.remove(addressList.get(n));
                                                                twoWaySet1.addFirst(addressList.get(n));
                                                        }

                                                        //Print out content of set 1
                                                        System.out.println("Recently accessed block addresses in set 1: Mem["
                                                                        + Integer.parseInt((String)twoWaySet1.peekFirst(), 2) + "], Mem["
                                                                        + Integer.parseInt((String)twoWaySet1.peekLast(), 2) + "]");
                                                }
                                        }
                                        else if(n % 4 == 2){
                                                if(twoWaySet2.size() <= 1)
                                                        System.out.println("Recently accessed block addresses in set 2: Mem["
                                                                        + Integer.parseInt((String)twoWaySet2.peekFirst(), 2) + "]");
                                                else {
                                                        if(n == Integer.parseInt((String)twoWaySet2.peekFirst(), 2)){//accessed address
matches most recent value.

                                                                //Do nothing
                                                        }
                                                        else if(n == Integer.parseInt((String)twoWaySet2.peekLast(), 2)){//accessed address
matches the LRU value.

                                                                twoWaySet2.remove(addressList.get(n));
                                                                twoWaySet2.addFirst(addressList.get(n));
                                                        }

                                                        //Print out content of set 2
                                                        System.out.println("Recently accessed block addresses in set 2: Mem["
                                                                        + Integer.parseInt((String)twoWaySet2.peekFirst(), 2) + "], Mem["
                                                                        + Integer.parseInt((String)twoWaySet2.peekLast(), 2) + "]");
                                                }
                                        }
                                        else if(n % 4 == 3){
                                                if(twoWaySet3.size() <= 1)
                                                        System.out.println("Recently accessed block addresses in set 3: Mem["
                                                                        + Integer.parseInt((String)twoWaySet3.peekFirst(), 2) + "]");
                                                else {
                                                        if(n == Integer.parseInt((String)twoWaySet3.peekFirst(), 2)){//accessed address
matches most recent value.

                                                                //Do nothing
                                                        }
                                                        else if(n == Integer.parseInt((String)twoWaySet3.peekLast(), 2)){//accessed address
matches the LRU value.

                                                                twoWaySet3.remove(addressList.get(n));
                                                                twoWaySet3.addFirst(addressList.get(n));
                                                        }

                                                        //Print out content of set 3
                                                        System.out.println("Recently accessed block addresses in set 3: Mem["
                                                                        + Integer.parseInt((String)twoWaySet3.peekFirst(), 2) + "], Mem["
                                                                        + Integer.parseInt((String)twoWaySet3.peekLast(), 2) + "]");
                                                }
                                        }
                                }
                                else {//miss branch
                                // Store recently accessed address in a set. If set is full, drop LRU to make room. Then print out content of
that set
                                        if(n % 4 == 0) {
                                                twoWaySet0.addFirst(addressList.get(n));
                                                if(twoWaySet0.size() > 2) {
                                                        twoWaySet0.removeLast();
                                                }
                                                //Print out the content of set 0
                                                if(twoWaySet0.size() <= 1)
                                                        System.out.println("Recently accessed block addresses in set 0: Mem["
                                                                        + Integer.parseInt((String)twoWaySet0.peekFirst(), 2) + "]");
                                                else
                                                        System.out.println("Recently accessed block addresses in set 0: Mem["
                                                                        + Integer.parseInt((String)twoWaySet0.peekFirst(), 2) + "], Mem["
                                                                        + Integer.parseInt((String)twoWaySet0.peekLast(), 2) + "]");
```

```
                        }
                        else if(n % 4 == 1) {
                                twoWaySet1.addFirst(addressList.get(n));
                                if(twoWaySet1.size() > 2) {
                                        twoWaySet1.removeLast();
                                }
                                //Print out the content of set 1
                                if(twoWaySet1.size() <= 1)
                                        System.out.println("Recently accessed block addresses in set 1: Mem["
                                                        + Integer.parseInt((String)twoWaySet1.peekFirst(), 2) + "]");
                                else
                                        System.out.println("Recently accessed block addresses in set 1: Mem["
                                                        + Integer.parseInt((String)twoWaySet1.peekFirst(), 2) + "], Mem["
                                                        + Integer.parseInt((String)twoWaySet1.peekLast(), 2) + "]");
                        }
                        else if(n % 4 == 2) {
                                twoWaySet2.addFirst(addressList.get(n));
                                if(twoWaySet2.size() > 2) {
                                        twoWaySet2.removeLast();
                                }
                                //Print out the content of set 2
                                if(twoWaySet2.size() <= 1)
                                        System.out.println("Recently accessed block addresses in set 2: Mem["
                                                        + Integer.parseInt((String)twoWaySet2.peekFirst(), 2) + "]");
                                else
                                        System.out.println("Recently accessed block addresses in set 2: Mem["
                                                        + Integer.parseInt((String)twoWaySet2.peekFirst(), 2) + "], Mem["
                                                        + Integer.parseInt((String)twoWaySet2.peekLast(), 2) + "]");
                        }
                        else if(n % 4 == 3) {
                                twoWaySet3.addFirst(addressList.get(n));
                                if(twoWaySet3.size() > 2) {
                                        twoWaySet3.removeLast();
                                }
                                //Print out the content of set 3
                                if(twoWaySet3.size() <= 1)
                                        System.out.println("Recently accessed block addresses in set 3: Mem["
                                                        + Integer.parseInt((String)twoWaySet3.peekFirst(), 2) + "]");
                                else
                                        System.out.println("Recently accessed block addresses in set 3: Mem["
                                                        + Integer.parseInt((String)twoWaySet3.peekFirst(), 2) + "], Mem["
                                                        + Integer.parseInt((String)twoWaySet3.peekLast(), 2) + "]");
                        }
                }
        }
}

LinkedList fourWaySet0 = new LinkedList();
LinkedList fourWaySet1 = new LinkedList();
LinkedList fourWaySet2 = new LinkedList();
LinkedList fourWaySet3 = new LinkedList();
//**4 way set associative mapping**
if(associative == 3) {
        System.out.println("4-way associative selected");
        Boolean hit = false;
        int n = 0;
        while(n >= 0 && n <= 15) {
                n = input.nextInt();
                hit = false;
                if(n < 0 || n > 15)
                        break;
                else {//check for match. If match found, hit. Otherwise, miss.
                        if(n % 4 == 0) {
                                for(int i = 0; i < fourWaySet0.size(); i++) {
                                        if(n == Integer.parseInt((String)fourWaySet0.get(i), 2)) {
                                                hit = true;
                                        }
                                }
                        }
                        else if(n % 4 == 1) {
                                for(int i = 0; i < fourWaySet1.size(); i++) {
                                        if(n == Integer.parseInt((String)fourWaySet1.get(i), 2)) {
                                                hit = true;
                                        }
                                }
                        }
                        else if(n % 4 == 2) {
                                for(int i = 0; i < fourWaySet2.size(); i++) {
                                        if(n == Integer.parseInt((String)fourWaySet2.get(i), 2)) {
                                                hit = true;
                                        }
                                }
                        }
                        else if(n % 4 == 3) {
                                for(int i = 0; i < fourWaySet3.size(); i++) {
                                        if(n == Integer.parseInt((String)fourWaySet3.get(i), 2)) {
                                                hit = true;
```

```
                            }
                        }
                    }
                    count += 1;

                    if(hit) {//hit branch
                        System.out.println("hit");

                        if(n % 4 == 0) {
                            if(n == Integer.parseInt((String)fourWaySet0.getFirst(), 2)){
                                //do nothing. Matched address number is already most recently accessed in the
cache
                            }
                            else {//Matched address number is not the most recently accessed in the cache. Move it
to the front of list.
                                for(int i = 1; i < fourWaySet0.size(); i++) {
                                    if(n == Integer.parseInt((String)fourWaySet0.get(i), 2)) {
                                        fourWaySet0.remove(i);
                                        fourWaySet0.addFirst(addressList.get(n));
                                        break;
                                    }
                                }
                            }
                            //Print out content of set 0
                            System.out.print("Recently accessed block addresses in set 0: ");
                            for(int i = 0; i < fourWaySet0.size(); i++) {
                                System.out.print("Mem[" + Integer.parseInt((String)fourWaySet0.get(i), 2) + "]
");
                            }
                        }
                        else if(n % 4 == 1) {
                            if(n == Integer.parseInt((String)fourWaySet1.getFirst(), 2)){
                                //do nothing. Matched address number is already most recently accessed in the
cache
                            }
                            else {//Matched address number is not the most recently accessed in the cache. Move it
to the front of list.
                                for(int i = 1; i < fourWaySet1.size(); i++) {
                                    if(n == Integer.parseInt((String)fourWaySet1.get(i), 2)) {
                                        fourWaySet1.remove(i);
                                        fourWaySet1.addFirst(addressList.get(n));
                                        break;
                                    }
                                }
                            }
                            //Print out content of set 1
                            System.out.print("Recently accessed block addresses in set 1: ");
                            for(int i = 0; i < fourWaySet1.size(); i++) {
                                System.out.print("Mem[" + Integer.parseInt((String)fourWaySet1.get(i), 2) + "]
");
                            }
                        }
                        else if(n % 4 == 2) {
                            if(n == Integer.parseInt((String)fourWaySet2.getFirst(), 2)){
                                //do nothing. Matched address number is already most recently accessed in the
cache
                            }
                            else {//Matched address number is not the most recently accessed in the cache. Move it
to the front of list.
                                for(int i = 1; i < fourWaySet2.size(); i++) {
                                    if(n == Integer.parseInt((String)fourWaySet2.get(i), 2)) {
                                        fourWaySet2.remove(i);
                                        fourWaySet2.addFirst(addressList.get(n));
                                        break;
                                    }
                                }
                            }
                            //Print out content of set 2
                            System.out.print("Recently accessed block addresses in set 2: ");
                            for(int i = 0; i < fourWaySet2.size(); i++) {
                                System.out.print("Mem[" + Integer.parseInt((String)fourWaySet2.get(i), 2) + "]
");
                            }
                        }
                        else if(n % 4 == 3) {
                            if(n == Integer.parseInt((String)fourWaySet3.getFirst(), 2)){
                                //do nothing. Matched address number is already most recently accessed in the
cache
                            }
                            else {//Matched address number is not the most recently accessed in the cache. Move it
to the front of list.
                                for(int i = 1; i < fourWaySet3.size(); i++) {
                                    if(n == Integer.parseInt((String)fourWaySet3.get(i), 2)) {
                                        fourWaySet3.remove(i);
                                        fourWaySet3.addFirst(addressList.get(n));
                                        break;
                                    }
                                }
                            }
                            //Print out content of set 3
                            System.out.print("Recently accessed block addresses in set 3: ");
                            for(int i = 0; i < fourWaySet3.size(); i++) {
                                System.out.print("Mem[" + Integer.parseInt((String)fourWaySet3.get(i), 2) + "]
");

                            }
                        }
                        System.out.println();
                    }
                    else { /*Miss branch.
                            *If set is full, drop LRU before adding new address in the front of the set.
                            *Print content of affected set.
                            */
                        System.out.println("miss");
                        countMiss += 1;

                        if(n % 4 == 0) {//Set 0 is full
                            fourWaySet0.addFirst(addressList.get(n));
                            if(fourWaySet0.size() > 4) {
                                fourWaySet0.removeLast();
                                System.out.print("Recently accessed block addresses in set 0: ");
                                for(int i = 0; i < fourWaySet0.size(); i++) {
                                    System.out.print("Mem[" + Integer.parseInt((String)fourWaySet0.get(i),
2) + "] ");

                                }
                            }
                            else {//Set 0 is not full
                                System.out.print("Recently accessed block addresses in set 0: ");
                                for(int i = 0; i < fourWaySet0.size(); i++) {
                                    System.out.print("Mem[" + Integer.parseInt((String)fourWaySet0.get(i),
2) + "] ");

                                }
                            }
                        }
                        else if(n % 4 == 1) {//Set 1 is full
                            fourWaySet1.addFirst(addressList.get(n));
                            if(fourWaySet1.size() > 4) {
                                fourWaySet1.removeLast();
                                System.out.print("Recently accessed block addresses in set 1: ");
                                for(int i = 0; i < fourWaySet1.size(); i++) {
                                    System.out.print("Mem[" + Integer.parseInt((String)fourWaySet1.get(i),
2) + "] ");

                                }
                            }
                            else {//Set 1 is not full
                                System.out.print("Recently accessed block addresses in set 1: ");
                                for(int i = 0; i < fourWaySet1.size(); i++) {
                                    System.out.print("Mem[" + Integer.parseInt((String)fourWaySet1.get(i),
2) + "] ");

                                }
                            }
                        }
                        else if(n % 4 == 2) {//Set 2 is full
                            fourWaySet2.addFirst(addressList.get(n));
                            if(fourWaySet2.size() > 4) {
                                fourWaySet2.removeLast();
                                System.out.print("Recently accessed block addresses in set 2: ");
                                for(int i = 0; i < fourWaySet2.size(); i++) {
                                    System.out.print("Mem[" + Integer.parseInt((String)fourWaySet2.get(i),
2) + "] ");

                                }
                            }
                            else {//Set 2 is not full
                                System.out.print("Recently accessed block addresses in set 2: ");
                                for(int i = 0; i < fourWaySet2.size(); i++) {
                                    System.out.print("Mem[" + Integer.parseInt((String)fourWaySet2.get(i),
2) + "] ");

                                }
                            }
                        }
                        else if(n % 4 == 3) {//Set 3 is full
                            fourWaySet3.addFirst(addressList.get(n));
                            if(fourWaySet3.size() > 4) {
                                fourWaySet3.removeLast();
                                System.out.print("Recently accessed block addresses in set 3: ");
                                for(int i = 0; i < fourWaySet3.size(); i++) {
                                    System.out.print("Mem[" + Integer.parseInt((String)fourWaySet3.get(i),
2) + "] ");

                                }
                            }
                            else {//Set 3 is not full
                                System.out.print("Recently accessed block addresses in set 3: ");
                                for(int i = 0; i < fourWaySet3.size(); i++) {
                                    System.out.print("Mem[" + Integer.parseInt((String)fourWaySet3.get(i),
2) + "] ");
```

```
                                                }
                                        }
                                }
                                System.out.println();
                        }
                }
        }
}

LinkedList fullSet = new LinkedList();
//**full associative mapping**
if(associative == 4) {
        System.out.println("full associative selected");
        Boolean hit = false;
        int n = 0;

        while(n >= 0 && n <= 15) {
                n = input.nextInt();
                hit = false;
                if(n < 0 || n > 15)
                        break;//end the loop
                else {
                        //Search for n in the cache. If match found, hit. Otherwise, miss.
                        for(int i = 0; i < fullSet.size(); i++) {
                                if(n == Integer.parseInt((String)fullSet.get(i), 2)) {
                                        hit = true;
                                }
                        }
                        count += 1;
                }
                if(hit) {//hit branch
                        System.out.println("hit");
                        if(n == Integer.parseInt((String)fullSet.getFirst(), 2)) {
                                //Do nothing. Matched value is already most recently used
                        }
                        else {//Matched value is not most recently used. Move value to the front of the list.
                                for(int i = 1; i < fullSet.size(); i++) {
                                        if(n == Integer.parseInt((String)fullSet.get(i), 2)) {
                                                fullSet.remove(i);
                                                fullSet.addFirst(addressList.get(n));
                                        }
                                }
                        }
                        System.out.print("Recently accessed block addresses in cache: ");
                        for(int i = 0; i < fullSet.size(); i++)
                                System.out.print("Mem[" + Integer.parseInt((String)fullSet.get(i), 2) + "] ");
                }
                else {//miss branch
                        System.out.println("miss");
                        countMiss += 1;
                        fullSet.addFirst(addressList.get(n));

                        if(fullSet.size() > 4) {//cache is full
                                fullSet.removeLast();
                                System.out.print("Recently accessed block addresses in cache: ");
                                for(int i = 0; i < fullSet.size(); i++)
                                        System.out.print("Mem[" + Integer.parseInt((String)fullSet.get(i), 2) + "] ");
                        }
                        else { //cache is not full
                                System.out.print("Recently accessed block addresses in cache: ");
                        for(int i = 0; i < fullSet.size(); i++)
                                System.out.print("Mem[" + Integer.parseInt((String)fullSet.get(i), 2) + "] ");
                        }
                }
                System.out.println("\ninput the next address number to search");
        }
}

//invalid input
if(associative != 1 && associative != 2 && associative != 3 && associative != 4) {
        System.out.println("Invalid input. Terminating program");
        System.exit(0);
}
input.close();

//**Nearing the end of the program**
System.out.println("Integer value outside of numbers 0-15 inputted. Ending the program.");
//Print out results and content of the cache when an integer outside of 0-15 is entered
missRate = ((countMiss / count) * 100);
System.out.println("***Results***");

//Stats (total attempts, # of hits and misses, and miss rate)
System.out.println("**Stats**");
System.out.println("Total attempts: " + (int)count);
System.out.println("Hits: " + (int)(count - countMiss));
System.out.println("Misses: " + (int)countMiss);
if(countMiss == 0)
```

```java
                        System.out.println("Miss rate: 0%");
                else
                        System.out.println("Miss rate: " + missRate + "%");

                //Contents of cache based on which cache mode was selected.
                System.out.println("**Content of cache**");
                if(associative == 1) {
                        System.out.println("Direct mapped cache");
                        System.out.println("cache0: Mem[" + Integer.parseInt((String)cache0.peekFirst(), 2) + "]");
                        System.out.println("cache1: Mem[" + Integer.parseInt((String)cache1.peekFirst(), 2) + "]");
                        System.out.println("cache2: Mem[" + Integer.parseInt((String)cache2.peekFirst(), 2) + "]");
                        System.out.println("cache3: Mem[" + Integer.parseInt((String)cache3.peekFirst(), 2) + "]");
                }
                if(associative == 2) {
                        System.out.println("2-way associative cache");
                        if(twoWaySet0.size() < 2)
                                System.out.println("set 0: Mem[" + Integer.parseInt((String)twoWaySet0.peekFirst(), 2) + "]");
                        else
                                System.out.println("set 0: Mem[" + Integer.parseInt((String)twoWaySet0.peekFirst(), 2)
                                + "] Mem[" + Integer.parseInt((String)twoWaySet0.peekLast(), 2) + "]");
                        if(twoWaySet1.size() < 2)
                                System.out.println("set 1: Mem[" + Integer.parseInt((String)twoWaySet1.peekFirst(), 2) + "]");
                        else
                                System.out.println("set 1: Mem[" + Integer.parseInt((String)twoWaySet1.peekFirst(), 2)
                                + "] Mem[" + Integer.parseInt((String)twoWaySet1.peekLast(), 2) + "]");
                        if(twoWaySet2.size() < 2)
                                System.out.println("set 2: Mem[" + Integer.parseInt((String)twoWaySet2.peekFirst(), 2) + "]");
                        else
                                System.out.println("set 2: Mem[" + Integer.parseInt((String)twoWaySet2.peekFirst(), 2)
                                + "] Mem[" + Integer.parseInt((String)twoWaySet2.peekLast(), 2) + "]");
                        if(twoWaySet3.size() < 2)
                                System.out.println("set 3: Mem[" + Integer.parseInt((String)twoWaySet3.peekFirst(), 2) + "]");
                        else
                                System.out.println("set 3: Mem[" + Integer.parseInt((String)twoWaySet3.peekFirst(), 2)
                                + "] Mem[" + Integer.parseInt((String)twoWaySet3.peekLast(), 2) + "]");
                }
                if(associative == 3) {
                        System.out.println("4-way associative cache");
                        System.out.print("set 0: ");
                        for(int i = 0; i < fourWaySet0.size(); i++) {
                                System.out.print("Mem[" + Integer.parseInt((String)fourWaySet0.get(i), 2) + "] ");
                        }
                        System.out.print("\nset 1:");
                        for(int i = 0; i < fourWaySet1.size(); i++) {
                                System.out.print("Mem[" + Integer.parseInt((String)fourWaySet1.get(i), 2) + "] ");
                        }
                        System.out.print("\nset 2:");
                        for(int i = 0; i < fourWaySet2.size(); i++) {
                                System.out.print("Mem[" + Integer.parseInt((String)fourWaySet2.get(i), 2) + "] ");
                        }
                        System.out.print("\nset 3:");
                        for(int i = 0; i < fourWaySet3.size(); i++) {
                                System.out.print("Mem[" + Integer.parseInt((String)fourWaySet3.get(i), 2) + "] ");
                        }
                }
                if(associative == 4) {
                        System.out.println("Fully associative cache");
                        System.out.print("Cache: ");
                        for(int i = 0; i < fullSet.size(); i++)
                                System.out.print("Mem[" + Integer.parseInt((String)fullSet.get(i), 2) + "] ");
                }
                System.exit(0);
        }
}
```