**IEEE**

import java.util.Scanner;

import javax.swing.text.Position.Bias;

public class IEEE {

    static String inttoBinary(int n){

        StringBuilder sb = new StringBuilder();

        if(n==0){

            sb.append("0");

        }else{

            while(n>0){

                sb.append(n%2);

                n/=2;

            }

            sb.reverse();

        }

        return sb.toString();

    }

    static String fracToBinary(float frac,int limit){

        StringBuilder sb = new StringBuilder();

        while(frac>0 && sb.length()<limit){

            frac \*=2;

            if (frac>=1){

                sb.append("1");

                frac -=1;

            }else{

                sb.append("0");

            }

        }

        return sb.toString();

    }

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter a decimal number:");

        float num = sc.nextFloat();

        int bits = Float.floatToIntBits(num);

        int sign = (bits>>>31) & 1;

        int exponent = (bits>>>23) & 0xFF;

        int mantissa = bits & 0x7FFFFF;

        int intPart = (int)num;

        float fracPart = num - intPart;

        String intBinary = inttoBinary(intPart);

        String fracBinary = fracToBinary(fracPart, 10);

        String FullBinary = intBinary+"."+fracPart;

        int shift = intBinary.length()-1;

        int realExp = shift;

        int biasedExp = realExp + 127;

        String BiasExpBinary = String.format("%8s",Integer.toBinaryString(biasedExp));

        String mantissastr = (intBinary.substring(1)+fracBinary);

        mantissastr = String.format("%23s",mantissastr).replace(' ','0');

        String bin32 = String.format("%1s %8s %23s",sign,BiasExpBinary,mantissastr);

        String hex = String.format("%08x",bits);

        System.out.println("Binary :"+FullBinary);

        System.out.println("real exponent :"+realExp);

        System.out.println("32-bit IEEE 754 representation:");

        System.out.println("Biased Exponent :"+biasedExp+" -> "+BiasExpBinary);

        System.out.println("Mantissa :"+mantissastr);

        System.out.println("Sign bit :"+sign);

        System.out.println("Final 32-bit representation :"+bin32);

        System.out.println("Hexadecimal representation :"+hex.);

        sc.close();

    }

}

**Mapping**

**Direct :** import java.util.Scanner;

public class DirectMapping {

    static int log2(int x) {

        return (int) (Math.log(x) / Math.log(2));

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Size of Cache memory (in KB): ");

        int cacheKB = sc.nextInt();

        int cacheSize = cacheKB \* 1024;

        System.out.print("Size of Main memory (in MB): ");

        int mainMB = sc.nextInt();

        int mainSize = mainMB \* 1024 \* 1024;

        int addrBits = log2(mainSize);

        System.out.println("Main memory address = " + addrBits + " bits");

        System.out.print("Size of each cache line (in Bytes): ");

        int lineSize = sc.nextInt();

        int numCacheLines = cacheSize / lineSize;

        int numBlocks = mainSize / lineSize;

        System.out.println("\n=== Results ===");

        System.out.println("Select cache mapping policy: Direct mapping");

        System.out.println("Size of Cache memory = " + cacheKB + " KB");

        System.out.println("Size of Main memory = " + mainMB + " MB");

        System.out.println("Main memory address = " + addrBits + " bits");

        System.out.println("Size of each cache line = " + lineSize + " Bytes");

        System.out.println("Number of cache banks = 1");

        System.out.println("Hence, size of cache bank = " + cacheKB + " KB");

        System.out.println("Cache lines per cache bank = " + cacheSize + "/" + lineSize +

                " = " + numCacheLines + " (Line No-0 to Line No-" + (numCacheLines - 1) + ")");

        System.out.println("Number of main memory blocks = " + mainSize + "/" + lineSize +

                " = " + numBlocks + " (Block -0 to Block No-" + (numBlocks - 1) + ")");

        int byteBits = log2(lineSize);

        int lineBits = log2(numCacheLines);

        int tagBits = addrBits - (lineBits + byteBits);

        System.out.println("Main memory address of " + addrBits + " bits is interpreted in 3 fields as calculated below:");

        System.out.println("LSB " + byteBits + " bits for Byte selection");

        System.out.println("Middle " + lineBits + " bits for Cache line selection");

        System.out.println("MSB " + tagBits + " bits (remaining) for the Tags");

        System.out.print("Input any Main memory block number for cache mapping = ");

        int blockNum = sc.nextInt();

        int lineNum = blockNum % numCacheLines;

        System.out.println("Block " + blockNum + " is mapped into cache line number = " + lineNum);

        sc.close();

    }

}

**2.way**

import java.util.Scanner;

public class TwoWaySetAss {

    static int log2(int x) {

        return (int) (Math.log(x) / Math.log(2));

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Size of Cache memory (in KB): ");

        int cacheKB = sc.nextInt();

        int cacheSize = cacheKB \* 1024;

        System.out.print("Size of Main memory (in MB): ");

        int mainMB = sc.nextInt();

        int mainSize = mainMB \* 1024 \* 1024;

        int addrBits = log2(mainSize);

        System.out.println("Main memory address = " + addrBits + " bits");

        System.out.print("Size of each cache line (in Bytes): ");

        int lineSize = sc.nextInt();

        int numCacheLines = cacheSize / lineSize;

        int numBlocks = mainSize / lineSize;

        int sets = numCacheLines / 2;

        System.out.println("\n=== Results ===");

        System.out.println("Select cache mapping policy: 2-Way Set Associative mapping");

        System.out.println("Size of Cache memory = " + cacheKB + " KB");

        System.out.println("Size of Main memory = " + mainMB + " MB");

        System.out.println("Main memory address = " + addrBits + " bits");

        System.out.println("Size of each cache line = " + lineSize + " Bytes");

        System.out.println("Number of sets = " + sets + " (Each set has 2 lines)");

        System.out.println("Cache lines per set = 2");

        System.out.println("Number of main memory blocks = " + numBlocks);

        int byteBits = log2(lineSize);

        int setBits = log2(sets);

        int tagBits = addrBits - (setBits + byteBits);

        System.out.println("Main memory address of " + addrBits + " bits is interpreted in 3 fields as:");

        System.out.println("LSB " + byteBits + " bits for Byte selection");

        System.out.println("Middle " + setBits + " bits for Set selection");

        System.out.println("MSB " + tagBits + " bits for Tags");

        System.out.print("Input any Main memory block number for cache mapping = ");

        int blockNum = sc.nextInt();

        int setNum = blockNum % sets;

        System.out.println("Block " + blockNum + " is mapped into Set number = " + setNum);

        sc.close();

    }

}

**3. fullassociative:**

import java.util.Scanner;

public class FullyAssociative {

    static int log2(int x) {

        return (int) (Math.log(x) / Math.log(2));

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Size of Cache memory (in KB): ");

        int cacheKB = sc.nextInt();

        int cacheSize = cacheKB \* 1024;

        System.out.print("Size of Main memory (in MB): ");

        int mainMB = sc.nextInt();

        int mainSize = mainMB \* 1024 \* 1024;

        int addrBits = log2(mainSize);

        System.out.println("Main memory address = " + addrBits + " bits");

        System.out.print("Size of each cache line (in Bytes): ");

        int lineSize = sc.nextInt();

        int numCacheLines = cacheSize / lineSize;

        int numBlocks = mainSize / lineSize;

        System.out.println("\n=== Results ===");

        System.out.println("Select cache mapping policy: Fully Associative mapping");

        System.out.println("Size of Cache memory = " + cacheKB + " KB");

        System.out.println("Size of Main memory = " + mainMB + " MB");

        System.out.println("Main memory address = " + addrBits + " bits");

        System.out.println("Size of each cache line = " + lineSize + " Bytes");

        System.out.println("Number of cache lines = " + numCacheLines);

        System.out.println("Number of main memory blocks = " + numBlocks);

        int byteBits = log2(lineSize);

        int tagBits = addrBits - byteBits;

        System.out.println("Main memory address of " + addrBits + " bits is interpreted in 2 fields as:");

        System.out.println("LSB " + byteBits + " bits for Byte selection");

        System.out.println("MSB " + tagBits + " bits for Tags");

        System.out.print("Input any Main memory block number for cache mapping = ");

        int blockNum = sc.nextInt();

        System.out.println("Block " + blockNum + " can be mapped into ANY of the " + numCacheLines + " cache lines");

        sc.close();

    }

}