GATE INSTITUTE OF TECHNOLOGY & MANAGEMENT SCIENCES

(Approved by AICTE , Affiliated to Sri Venkateswara University , TIRUPATI)

ICET CODE : GTMT

Master of Computer Applications

SEMESTER - I

Hall Ticket No: _____

Name of the Student: M. lahasi

Subject: computer organization

Year: I Semester: I Batch 2025 - 2026

GATE EDUCATIONAL INSTITUTIONS

DEPARTMENT OF COMPUTERS

YEAR I SEMESTER I

Reg No._____

CERTIFICATE

Certified that this is the bonafide record of practical work done in the laboratory by the candidate M. Lahari *during the academic year* 2025 - 2027 *subject* computer organization.

No. of Practical's Conducted 08 ***No. of Practical's Attended*** 08

LECTURER **HEAD OF THE DEPARTMENT**

Submitted for the Practical Examination held on_____

Examiners

1.

2.

INDEX

S.No.	Date	Name of the Experiment	Page No.	Marks Awarded	Remarks
1	25/10/25	Number conversions	1-10		
2	30/10/25	Arithmetic operations	11-19		
3	7/11/25	Absolute loader	20-26		
4	18/11/25	Relocating loader	27-32		
5	28/11/25	Register operations	33-40		
6	11/12/25	ASCII for characters	41-48		
7	26/12/25	logic gates	49-56		
8	03/01/26	Half adder and full adder	57-62		

1. Number conversions

Practical No. : 01

1. Write a Java Program to perform number conversions

Aim:- To demonstrate a Java Program to perform number conversions

Description :- This Program performs number system conversions using a menu-driven approach

The user can convert numbers between :

* Decimal ⟷ Binary
* Decimal ⟷ octal
* Binary ⟷ Hexa decimal
* Hexadecimal → Binary

The Program repeatedly displays a menu, takes the user's choice, performs the selected conv-ersion using separate functions, and displays the result.

Dynamic memory allocation is used where required, and the program terminates when the user selects the exit option.

Algorithm :-

Step 1 :- Start the program

Step 2 :- Display the menu with conversion options :-

1. Decimal to Binary
2. Binary to Decimal
3. Decimal to octal
4. octal to Decimal
5. Hexa decimal to Binary
6. Binary to Hexadecimal
7. Exit

Step 3 :- Read the user's choice

Step 4 :- If the choice is 7, display "Good bye" and stop the program

Step 5 :- Otherwise, Perform the selected conversion:

* Decimal to Binary
  * Repeatedly divide the decimal number by 2
  * Store remainders
  * Reverse the result to get binary number
* Binary to Decimal
  * Multiply each bit by powers of 2
  * Add the values to get decimal
* Decimal to octal
  * convert decimal to octal using base 8 - conversion.

* Octal to Decimal
  * Multiply each digit by powers of 8
    Add the values
* Hexadecimal to Binary
  * Convert hexadecimal to decimal
  * convert decimal to binary
* Binary to Hexadecimal
  * Group binary digits into sets of four
  * convert each group to its hexadecim -al equivalent.

Step 6 :- Display the converted result

Step 7 :- Repeat steps 2 to 6 until the user selects Exit.

Step 8 :- End the program

1. Write a program to perform number conversions using Java.

Practical No. :

```
import java.util.Scanner;

public class NumberSystemConversion {

  /* ----------------------------------------
    Decimal to Binary
  ---------------------------------------- */
  static String decimalToBinary(int decimal) {
    if (decimal == 0)
      return "0";

    StringBuilder binary = new StringBuilder();

    while (decimal > 0) {
      binary.append(decimal % 2);
      decimal /= 2;
    }

    return binary.reverse().toString();
  }

  /* ----------------------------------------
    Binary to Decimal
  ---------------------------------------- */
  static int binaryToDecimal(String binary) {
    int decimal = 0;
```

```
    for (int i = 0; i < binary.length(); i++) {
        decimal = decimal * 2 + (binary.charAt(i) - '0');
    }

    return decimal;
}

/* ----------------------------------------
   Decimal to Octal
---------------------------------------- */
static String decimalToOctal(int decimal) {
    return Integer.toOctalString(decimal);
}

/* ----------------------------------------
   Octal to Decimal
---------------------------------------- */
static int octalToDecimal(String octal) {
    int decimal = 0;

    for (int i = 0; i < octal.length(); i++) {
        decimal = decimal * 8 + (octal.charAt(i) - '0');
    }

    return decimal;
}
```

```
/* ----------------------------------------
   Hexadecimal to Binary
---------------------------------------- */
static String hexadecimalToBinary(String hex) {
    int decimal = Integer.parseInt(hex, 16);
    return decimalToBinary(decimal);
}

/* ----------------------------------------
   Binary to Hexadecimal
---------------------------------------- */
static String binaryToHexadecimal(String binary) {
    int decimal = binaryToDecimal(binary);
    return Integer.toHexString(decimal).toUpperCase();
}

/* ----------------------------------------
   Main Method
---------------------------------------- */
public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    int choice;

    while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Decimal to Binary");
        System.out.println("2. Binary to Decimal");
```

```
System.out.println("3. Decimal to Octal");
System.out.println("4. Octal to Decimal");
System.out.println("5. Hexadecimal to Binary");
System.out.println("6. Binary to Hexadecimal");
System.out.println("7. Exit");

System.out.print("Enter your choice: ");
choice = sc.nextInt();

if (choice == 7) {
  System.out.println("Goodbye!");
  break;
}

switch (choice) {

  case 1:
    System.out.print("Enter a decimal number: ");
    int dec1 = sc.nextInt();
    System.out.println("Decimal to Binary: " + decimalToBinary(dec1));
    break;

  case 2:
    System.out.print("Enter a binary number: ");
    String bin1 = sc.next();
    System.out.println("Binary to Decimal: " + binaryToDecimal(bin1));
    break;
```

```
        case 3:
            System.out.print("Enter a decimal number: ");
            int dec2 = sc.nextInt();
            System.out.println("Decimal to Octal: " + decimalToOctal(dec2));
            break;

        case 4:
            System.out.print("Enter an octal number: ");
            String oct = sc.next();
            System.out.println("Octal to Decimal: " + octalToDecimal(oct));
            break;

        case 5:
            System.out.print("Enter a hexadecimal number: ");
            String hex = sc.next();
            System.out.println("Hexadecimal to Binary: " + hexadecimalToBinary(hex));
            break;

        case 6:
            System.out.print("Enter a binary number: ");
            String bin2 = sc.next();
            System.out.println("Binary to Hexadecimal: " + binaryToHexadecimal(bin2));
            break;

        default:
            System.out.println("Invalid choice!");
    }
}
```

```
        sc.close();
    }
}
```

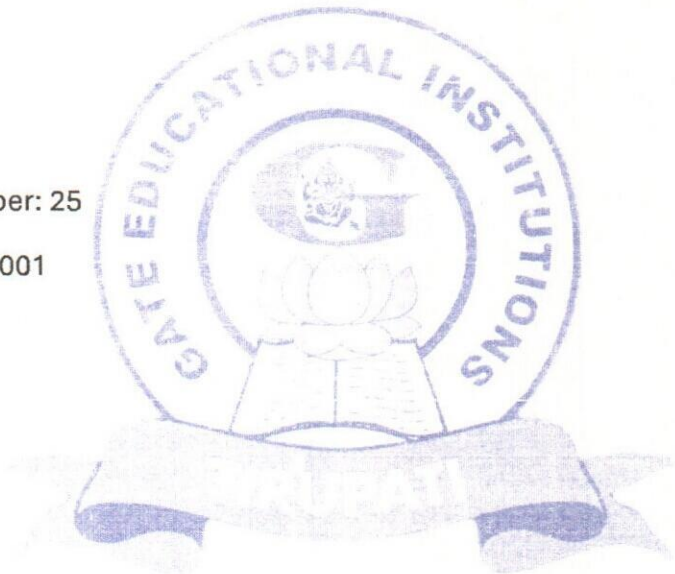
Menu:

1. Decimal to Binary
2. Binary to Decimal
3. Decimal to Octal
4. Octal to Decimal
5. Hexadecimal to Binary
6. Binary to Hexadecimal
7. Exit

Enter your choice: 1

Enter a decimal number: 25

Decimal to Binary: 11001



2. Arithmetic operations

2. write a Java Program to perform arithmetic operations

Aim :- To demonstrate a Java Program to perform arithmetic operations.

Description :- This Program performs binary arithmetic operations such as Addition, Subtraction, Multiplication, and Division without using arithmetic operators (+, -, *, /).

Instead, it uses bitwise operators like AND(&), XOR (^), NOT (~), and left shift (<<)

The Program is menu-driven, allowing the user to select an operation and enter two integers. The selected binary operation is performed and the result is displayed.

Algorithm :-

Step 1 :- Start the Program

Step 2 :- Display menu with options

1. Binary Addition
2. Binary Subtraction
3. Binary Multiplication
4. Binary Division
5. Exit

step 3:- Read the user's choice

step 4:- if choice is not exit

* Read two integers n_1 and n_2

step 5:- perform operation based on choice

case 1 : Binary Addition

1. Repeat until n_2 becomes 0
2. compute carry as $(n_1 \& n_2) << 1$
3. compute sum as $n_1 \wedge n_2$
4. Assign sum to n_1 and carry to n_2
5. Display result

case 2 : Binary subtraction

1. Find 2's complement of n_2
2. Add it to n_1 using binary addition
3. Display result.

case 3 : Binary Multiplication

1. Initialize result as 0
2. Add n_1 to result repeatedly n_2 times
3. Display result.

case 4 : Binary Division

1. If $n_1 < n_2$, result is 0
2. Subtract n_2 from n_1 repeatedly using binary addition

3. count number of subtractions

4. Display quotient.

Step6 :- Repeat steps until User selects Exit

Step 7 :- stop the program

2. **Write a Java program to perform arithmetic operations**

```
import java.util.Scanner;

public class BinaryOperations {

  /* performs binary addition for the given values */
  static int binaryAddition(int n1, int n2) {
    int carry;
    while (n2 != 0) {
      // calculating the carry and left shift
      carry = (n1 & n2) << 1;
      // calculating the sum
      n1 = n1 ^ n2;
      n2 = carry;
    }
    return n1;
  }

  /* performs binary subtraction for the given values */
  static int binarySubtraction(int n1, int n2) {
    int carry;
    // finding two's complement of n2
    n2 = binaryAddition(~n2, 1);

    while (n2 != 0) {
      carry = (n1 & n2) << 1;
      n1 = n1 ^ n2;
      n2 = carry;
```

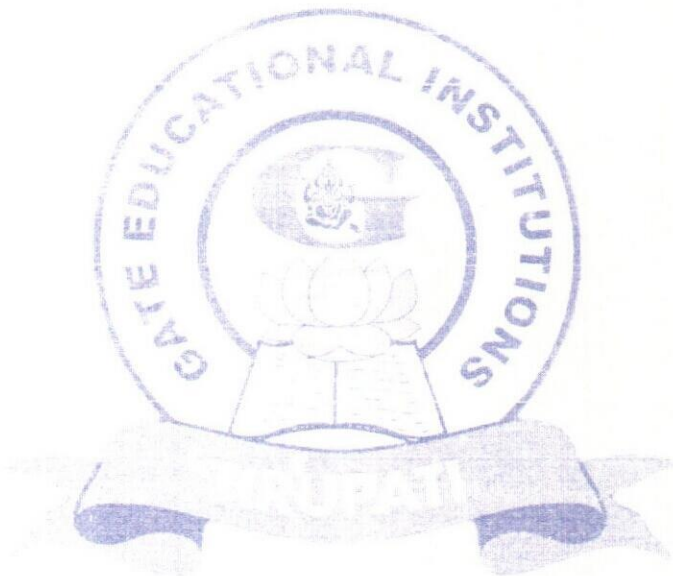
```
    }
    return n1;
}

/* performs binary multiplication for the given values */
static int binaryMultiplication(int n1, int n2) {
    int res = 0;
    for (int i = 0; i < n2; i++) {
        res = binaryAddition(res, n1);
    }
    return res;
}

/* performs binary division for the given values */
static int binaryDivision(int n1, int n2) {
    int res, count = 0, twosComplement;

    if (n1 < n2) {
        System.out.println("Division of " + n1 + " and " + n2 + " is 0");
        return 0;
    }

    res = n1;
    twosComplement = binaryAddition(~n2, 1);

    while (res > 0) {
        res = binaryAddition(res, twosComplement);
        if (res <= 0) {
```

```
        if (res == 0)
            count = binaryAddition(count, 1);
        break;
      } else {
        count = binaryAddition(count, 1);
      }
    }
    return count;
}

/* Main Method */
public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    int n1, n2, res, ch;

    while (true) {
      System.out.println("1. Binary Addition");
      System.out.println("2. Binary Subtraction");
      System.out.println("3. Binary Multiplication");
      System.out.println("4. Binary Division");
      System.out.println("5. Exit");
      System.out.print("Enter your choice: ");
      ch = sc.nextInt();

      if (ch == 5) {
        System.out.println("Exiting...");
        break;
```

```
}
```

```
System.out.print("Enter the value for n1 and n2: ");
n1 = sc.nextInt();
n2 = sc.nextInt();

switch (ch) {
  case 1:
    res = binaryAddition(n1, n2);
    break;

  case 2:
    res = binarySubtraction(n1, n2);
    break;

  case 3:
    res = binaryMultiplication(n1, n2);
    break;

  case 4:
    res = binaryDivision(n1, n2);
    break;

  default:
    System.out.println("Wrong option!!");
    continue;
}
```

```
        System.out.println("Result: " + res + "\n");
    }

    sc.close();
  }
}
```



OUTPUT:

1. Binary Addition

2. Binary Subtraction

3. Binary Multiplication

4. Binary Division

5. Exit

Enter your choice: 1

Enter the value for n1 and n2: 5 3

Result: 8

3. Absolute loader

3. Write a c program to implement an absolute loader

Aim :- TO implement an absolute loader in c

Description :- This program simulates a simple loader that reads an object program from an input file (input.dat) and produces a memory map in an output file (output.dat)

The object program contains :

* H (Header) record -> Starting address and program length
* T (Text) record -> address and object code
* E (End) record -> end of program

The program reads object code byte by byte, assigns consecutive memory addresses, and writes them into the output file until the End (E) record is encountered

Algorithm :

Step 1 : Start the program

Step 2 :- Declare variables and file pointers

* input -> stores records read from file
* Start, length -> header information
* address -> memory address
* fp1, fp2 -> input and output file pointers

Step 3 :- Open files

* Open input.dat in read mode
* Open output.dat in write mode

Step 4:- Read the first record from input file

Step 5:- Repeat until end record (E) is found

a) if record is Header (H)

1. Read starting address
2. Read Program length
3. Read next input record

b) If record is Text (T)

1. Read starting address of object code
2. Read object code
3. Split object code into bytes
4. Write each byte with its memory address to output file
5. Increment address
6. Read next input record

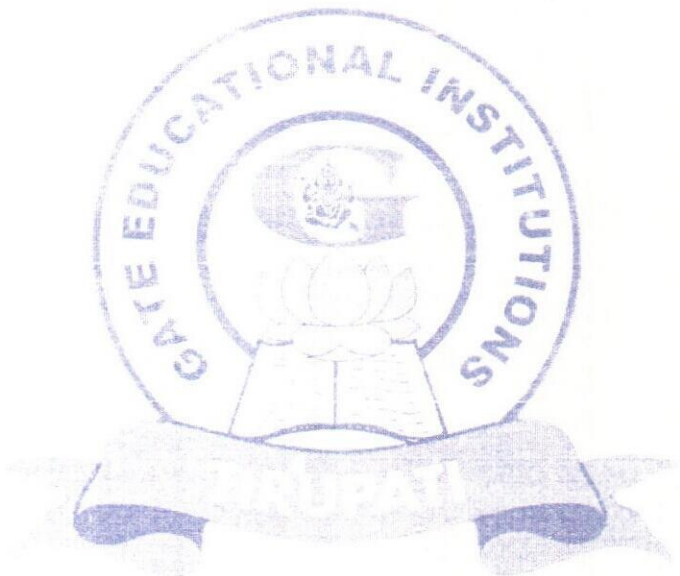
c) otherwise (remaining object code)

1. Write object code bytes to output file
2. Assign sequential addresses
3. Read next input record

Step 6:- Close input and output files

Step 7 :- Display "FINISHED"

Step 8 :- Stop the program.

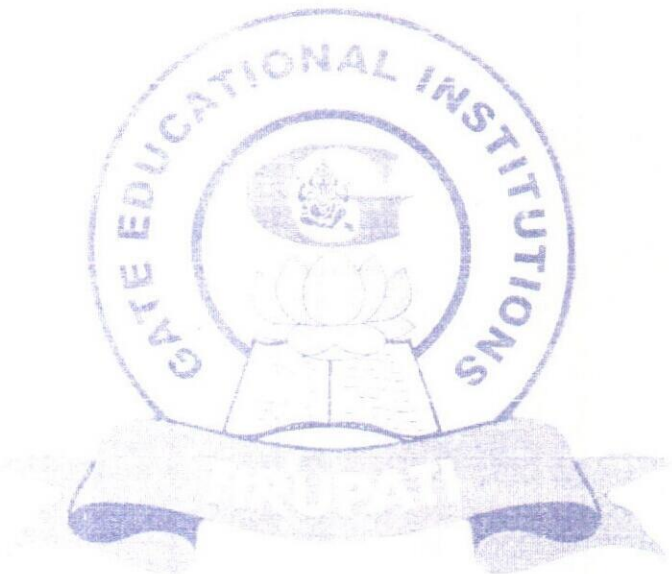


3. Implement an absolute loader in C.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
   char input[10];
   int start, length, address;
   FILE *fp1, *fp2;

   clrscr();

   /* Open input and output files */
   fp1 = fopen("input.dat", "r");
   fp2 = fopen("output.dat", "w");

   /* Read first record */
   fscanf(fp1, "%s", input);

   /* Continue until End record (E) */
   while (strcmp(input, "E") != 0)
   {
      /* Header record */
      if (strcmp(input, "H") == 0)
      {
         fscanf(fp1, "%d", &start);
         fscanf(fp1, "%d", &length);
```

```
        fscanf(fp1, "%s", input);
    }

    /* Text record with address */
    else if (strcmp(input, "T") == 0)
    {
        fscanf(fp1, "%d", &address);
        fscanf(fp1, "%s", input);

        fprintf(fp2, "%d\t%c%c\n", address,   input[0], input[1]);
        fprintf(fp2, "%d\t%c%c\n", address+1, input[2], input[3]);
        fprintf(fp2, "%d\t%c%c\n", address+2, input[4], input[5]);

        address += 3;
        fscanf(fp1, "%s", input);
    }

    /* Remaining object code */
    else
    {
        fprintf(fp2, "%d\t%c%c\n", address,   input[0], input[1]);
        fprintf(fp2, "%d\t%c%c\n", address+1, input[2], input[3]);
        fprintf(fp2, "%d\t%c%c\n", address+2, input[4], input[5]);

        address += 3;
        fscanf(fp1, "%s", input);
    }
}
```

```
    /* Close files */
    fclose(fp1);
    fclose(fp2);

    printf("FINISHED");
    getch();
}
```



Step 3 :- Read the actual starting address from the user

Step 4 :- Open files

*Open relin PUt.dat in read mode

*Open reloutPUt.dat in write mode

Step 5 :- Read the first record from input file

Step 6 :- Repeat until End record (E) is encountered

a) If record is Header (H)

1. Read Program name/address
2. Read Program length
3. Read next record

b) if record is Text (T)

1. Read Stasting address of text record
2. Read relocation bitmask
3. Add actual starting address to the text record address
4. Find length of bitmask
5. For each bit in the bitmask :

* Read opcode and address field

*If relocation bit = 0

*Actual address = address field

*If relocation bit = 1

*Actual address = address field + Starting address

INPUT FILE:

INPUT.DAT H 1000 232

T 1000 142033 483039 102036

T 2000 298300 230000 282030 302015 E

OUTPUT FILE:

OUTPUT.DAT 1000 14

1001 20

1002 33

1003 48

1004 30

1005 39

1006 10

1007 20

1008 36

2000 29

2001 83

2002 00

2003 23

2004 00

2005 00

2006 28

2007 20

2008 30

2009 30

2010 20

2011

4. write a c program to implement a relocating loader.

Aim :- To implement a relocating loader in c

Description :- This program implements a Relocating loader in c.

It reads a relocatable object program from an input file (relinput.dat) and produces a relocated Memory map in an output file (reloutput.dat).

The program uses a bitmask to determine wheater an address field in the object code needs relocation or not

If the relocation bit is 1, the actual starting address is added to the address field.

If the relocation bit is 0, the address remains unchanged.

The user provides the actual starting address at run-time

Algorithm :-

Step 1 :- Start the program

step 2 :- Declare variables and file pointers

* add, length, input -> store records
* bitmask -> relocation bits
* start -> actual starting address
* address, opcode, addr, actualadd -> Procesing variables

* write relocated address and object code to output file
* Increment address by instruction length

Step 6 :- Read next input record

Step 7 :- close input and output files

step 8 :- Display "FINISHED"

Step 9 :- Stop the program.

4. Implement a relocating loader in C.

Practical No. :

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

void main()
{
   char add[6], length[10], input[10];
   char binary[12], bitmask[12], relocbit;

   int start, inp, len, i;
   int address, opcode, addr, actualadd;

   FILE *fp1, *fp2;

   clrscr();

   /* Get actual starting address */
   printf("Enter the actual starting address : ");
   scanf("%d", &start);

   /* Open input and output files */
   fp1 = fopen("relinput.dat", "r");
   fp2 = fopen("reloutput.dat", "w");

   /* Read first record */
   fscanf(fp1, "%s", input);

   /* Continue until End record */
   while (strcmp(input, "E") != 0)
   {
      /* Header record */
      if (strcmp(input, "H") == 0)
      {
         fscanf(fp1, "%s", add);
         fscanf(fp1, "%s", length);
         fscanf(fp1, "%s", input);
      }

      /* Text record */
      if (strcmp(input, "T") == 0)
```

```
{
   fscanf(fp1, "%d", &address);
   fscanf(fp1, "%s", bitmask);

   /* Adjust starting address */
   address += start;

   len = strlen(bitmask);

   /* Process each relocation bit */
   for (i = 0; i < len; i++)
   {
      fscanf(fp1, "%d", &opcode);
      fscanf(fp1, "%d", &addr);

      relocbit = bitmask[i];

      if (relocbit == '0')
         actualadd = addr;
      else
         actualadd = addr + start;

      fprintf(fp2, "%d\t%d%d\n", address, opcode, actualadd);
      address += 3;
   }

   fscanf(fp1, "%s", input);
  }
 }

 /* Close files */
 fclose(fp1);
 fclose(fp2);

 printf("FINISHED");
 getch();
}
```

OUTPUT:

Enter the actual starting address :4000

RELOUTPUT.DAT 4000 144033

4003 484039

4006 901776

4009 921765

4012 574765

5011 234838

5014 434979

5017 894060

5020 664849

5023 991477

5. Register Operations

5. Write a program to perform register operations.

Aim :- To demonstrate a Java program to perform register operation.

Description :- This program simulates basic micro-operations performed on 8-bit CPU registers using the Java Programming language.

Two registers, R_1 and R_2 are represented using the Unit8_t data type to ensure 8-bit behaviour.

The program demonstrates common register-level micro-operations such as :

* Register Transfer
* Addition
* Bit wise AND
* Bit wise OR
* left shift
* Right shift.

A menu-driven interface allows the user to select an operation. After each operation, the contents of the registers are displayed in hexa decimal format.

Algorithm :-

Step1 :- start the program

Step2 :- Initialize registers

* Set $R_1 = 0X0F$
* Set $R_2 = 0XF0$

Step 3 :- Display menu repeatedly until exit is selected

Step 4 :- Read user choice

Step 5 :- Perform selected micro-operation

case 1 : Register Transfer

1. copy contents of R_2 into R_1
2. Display register values

case 2 : Addition

1. Add contents of R_1 and R_2
2. Store result in R_1
3. Display register values

case 3 : Bitwise AND

1. Perform AND operation between R_1 and R_2
2. Store Result in R_1
3. Display register values

case 4 : Bitwise OR

1. Perform OR operation between R_1 and R_2
2. Store result in R_1
3. Display register values

case 5 : left shift

1. Shift bits of R_1 left by one position
2. Store result in R_1
3. Display register values

case 6 : Right shift

1. Shift bits of R_1 right by one position
2. Store result in R_1

3. Display register values

case 7: Reset Registers

1. Reset R_1 and R_2 to initial values
2. Display register values

case 8: Exit

1. Terminate the program

Step 6 :- Stop the program

5. Write a program to perform register operations in Java.

```
import java.util.Scanner;

public class MicroOperations {

  // Simulate 8-bit registers
  static int R1 = 0x0F; // 00001111
  static int R2 = 0xF0; // 11110000

  static void showRegisters() {
    System.out.printf("R1: 0x%02X\tR2: 0x%02X\n", R1 & 0xFF, R2 & 0xFF);
  }

  static void transfer() {
    R1 = R2 & 0xFF;
    System.out.println("Performed: R1 ← R2");
    showRegisters();
  }

  static void add() {
    R1 = (R1 + R2) & 0xFF;
    System.out.println("Performed: R1 ← R1 + R2");
    showRegisters();
  }

  static void bitwiseAnd() {
    R1 = (R1 & R2) & 0xFF;
    System.out.println("Performed: R1 ← R1 AND R2");
    showRegisters();
```

```
static void bitwiseOr() {
    R1 = (R1 | R2) & 0xFF;
    System.out.println("Performed: R1 ← R1 OR R2");
    showRegisters();
}

static void leftShift() {
    R1 = (R1 << 1) & 0xFF;
    System.out.println("Performed: R1 ← R1 << 1 (Left Shift)");
    showRegisters();
}

static void rightShift() {
    R1 = (R1 >> 1) & 0xFF;
    System.out.println("Performed: R1 ← R1 >> 1 (Right Shift)");
    showRegisters();
}

public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    int choice;

    while (true) {
        System.out.println("\n--- Micro-Operations Menu ---");
        System.out.println("1. Register Transfer (R1 ← R2)");
        System.out.println("2. Addition (R1 ← R1 + R2)");
```

```
System.out.println("3. Bitwise AND (R1 ← R1 AND R2)");
System.out.println("4. Bitwise OR (R1 ← R1 OR R2)");
System.out.println("5. Left Shift (R1 << 1)");
System.out.println("6. Right Shift (R1 >> 1)");
System.out.println("7. Reset Registers");
System.out.println("8. Exit");
System.out.print("Choose an operation: ");

choice = sc.nextInt();

switch (choice) {
    case 1: transfer(); break;
    case 2: add(); break;
    case 3: bitwiseAnd(); break;
    case 4: bitwiseOr(); break;
    case 5: leftShift(); break;
    case 6: rightShift(); break;
    case 7:
        R1 = 0x0F;
        R2 = 0xF0;
        System.out.println("Registers reset to default values.");
        showRegisters();
        break;
    case 8:
        System.out.println("Exiting program.");
        sc.close();
        return;
    default:
        System.out.println("Invalid choice. Try again.");
```

```
        }
    }
  }
}
```


GATE EDUCATIONAL INSTITUTIONS

OUTPUT:

--- Micro-Operations Menu ---

1. Register Transfer (R1 ← R2)

2. Addition (R1 ← R1 + R2)

3. Bitwise AND (R1 ← R1 AND R2)

4. Bitwise OR (R1 ← R1 OR R2)

5. Left Shift (R1 << 1)

6. Right Shift (R1 >> 1)

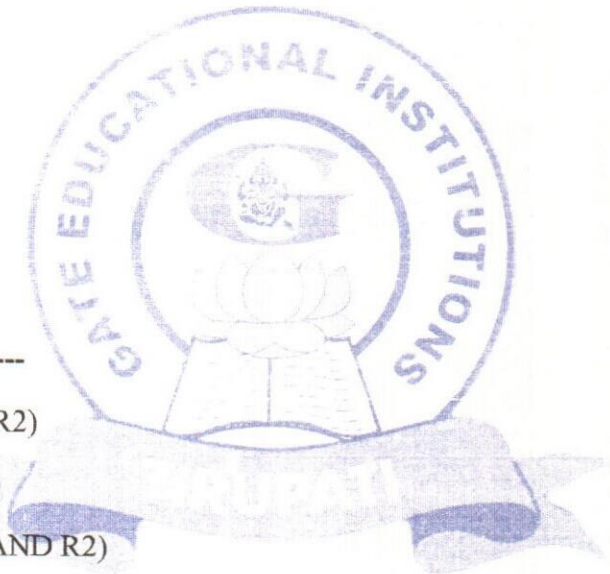
7. Reset Registers

8. Exit

Choose an operation: **2**

Performed: R1 ← R1 + R2

R1: 0xFF R2: 0xF0



--- Micro-Operations Menu ---

1. Register Transfer (R1 ← R2)

2. Addition (R1 ← R1 + R2)

3. Bitwise AND (R1 ← R1 AND R2)

4. Bitwise OR (R1 ← R1 OR R2)

5. Left Shift (R1 << 1)

6. Right Shift (R1 >> 1)

7. Reset Registers

8. Exit

Choose an operation: **8**

Exiting program.

6. Write a Java Program to Print ASCII for characters

Aim :- To demonstrate ASCII for characters in Java

Description :- This program displays the ASCII values of characters from 0 to 127.

ASCII (American standard code for information interchange) assigns a unique numeric value to each character.

The program uses a loop to iterate through all ASCII values and checks wheater a character is printable or non-printable

Printable characters (ASCII 32 to 126) are displayed as characters, while non-printable characters are labelled accordingly.

Algorithm :-

Step 1 :- Start the program

Step 2 :- Declare an Integer variable i;

Step 3 :- Display headings for ASCII table

Step 4 :- Use a for loop from i=0 to i=127

Step 5 :- For each value of i

1. Check if i is between 32 and 126

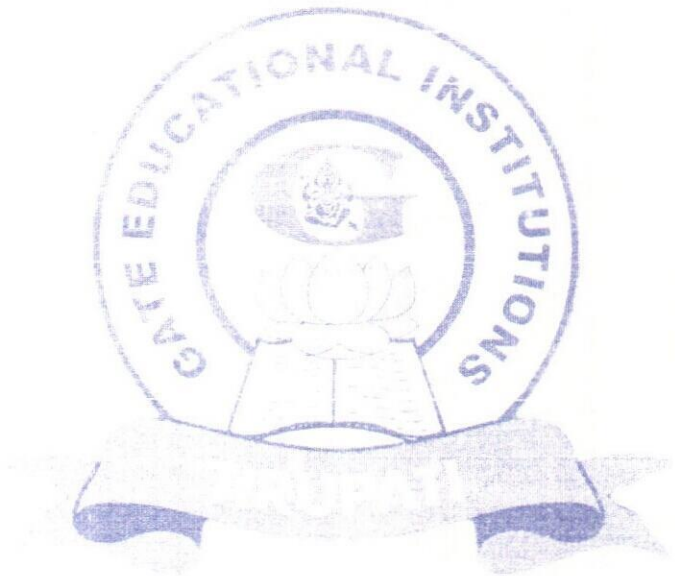
* If Yes, Print the character and its ASCII value

* If no, Print "Non-Printable" and the

ASCII value

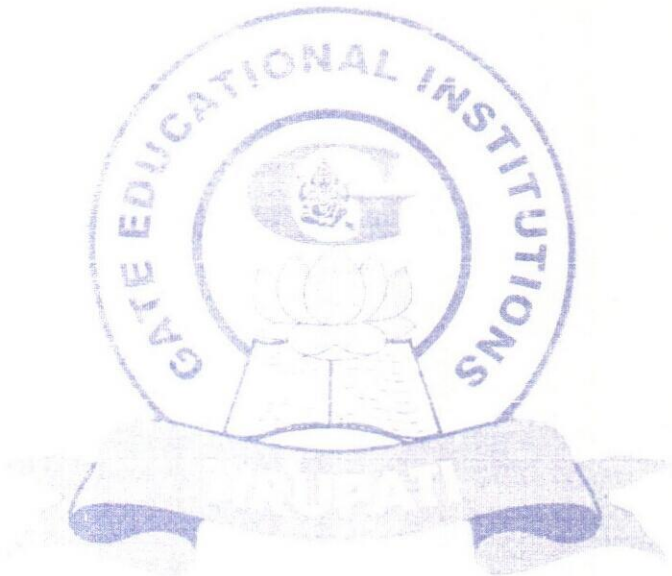
step 6: continue until all ASCII values are printed

step 7: Stop the program



6. Program to print ASCII for characters in Java.

```
public class ASCII {
   public static void main(String[] args) {
      for (int i = 0; i < 128; i++) {
         if (i >= 32 && i <= 126)
            System.out.println((char) i + " = " + i);
         else
            System.out.println("Non printable = " + i);
      }
   }
}
```



OUTPUT:

Non printable = 0

Non printable = 1

Non printable = 2

Non printable = 3

Non printable = 4

Non printable = 5

Non printable = 6

Non printable = 7

Non printable = 8

Non printable = 9

Non printable = 10

Non printable = 11

Non printable = 12

Non printable = 13

Non printable = 14

Non printable = 15

Non printable = 16

Non printable = 17

Non printable = 18

Non printable = 19

Non printable = 20

Non printable = 21

Non printable = 22

Non printable = 23

Non printable = 24

Non printable = 25

Non printable = 26

Non printable = 27

Non printable = 28

Non printable = 29

Non printable = 30

Non printable = 31

= 32

! = 33

" = 34

= 35

$ = 36

% = 37

& = 38

' = 39

(= 40

) = 41

* = 42

+ = 43

, = 44

- = 45

. = 46

/ = 47

0 = 48

1 = 49

2 = 50

3 = 51

4 = 52

5 = 53

6 = 54

7 = 55

8 = 56

9 = 57

: = 58

; = 59

< = 60

= = 61

> = 62

? = 63

@ = 64

A = 65

B = 66

C = 67

D = 68

E = 69

F = 70

G = 71

H = 72

I = 73

J = 74

K = 75

L = 76

M = 77

N = 78

O = 79

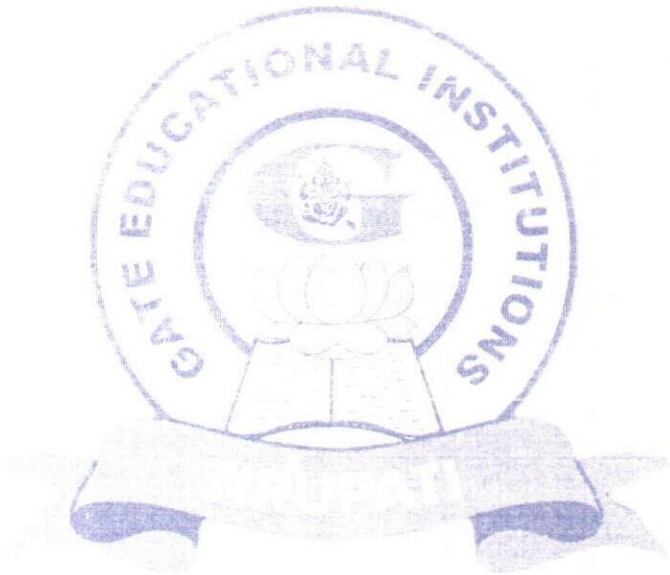
P = 80

Q = 81

R = 82

S = 83

T = 84



U = 85

V = 86

W = 87

X = 88

Y = 89

Z = 90

[= 91

\ = 92

] = 93

^ = 94

_ = 95

` = 96

a = 97

b = 98

c = 99

d = 100

e = 101

f = 102

g = 103

h = 104

i = 105

j = 106

k = 107

l = 108

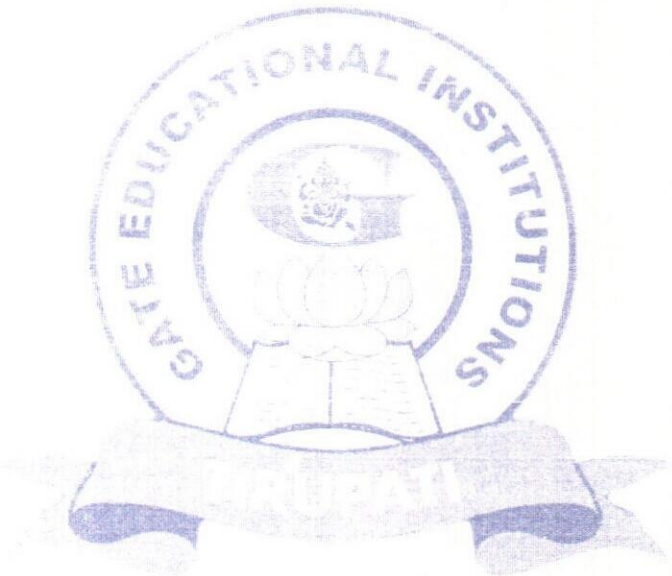
m = 109

n = 110

o = 111

p = 112

q = 113



r = 114

s = 115

t = 116

u = 117

v = 118

w = 119

x = 120

y = 121

z = 122

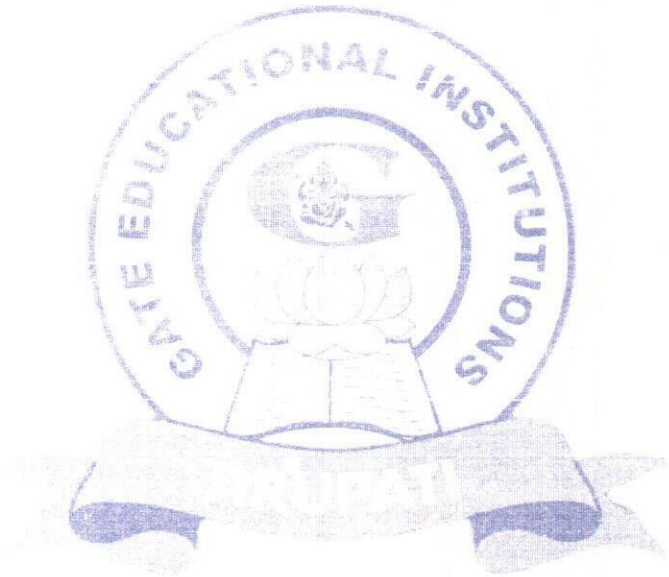
{ = 123

| = 124

} = 125

~ = 126

Non printable = 127



7. Write a Java Program to implement logic gates

Aim :- To implement logic gates using Java

Description :- This program implements basic digital logic gates using the Java programming language The logic gates included are AND, OR, NOT, XOR, NAND, NOR, and XNOR.

The program accepts binary inputs (0 or 1) from the user, performs the selected logic gate operation using bitwise and logical operators, and displays the result.

A menu-driven approach is used to allow the user to select the required logic gate.

Algorithm :-

Step 1 :- Start the program

Step 2 :- Display the list of logic gates

1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. XNOR

Step 3 : Read the user's choice

Step 4 :- Read input values

* If the selected gate is NOT, read only one input
* For all other gates, read two inputs

Step 5:- Validate inputs
* Check whether the inputs are either 0 or 1
* If invalid, display an error message and terminate

Step 6:- Perform the selected logic operation using switch-case.

case 1: AND Gate
output = a&b

case 2: OR Gate
output = a|b

case 3: NOT Gate
output = !a

case 4: XOR Gate
output = a^b

case 5: NAND Gate
output = !(a&b)

case 6: NOR Gate
output = !(a|b)

case 7: XNOR Gate
output = !(a^b)

Step 7:- Display the result

Step 8:- Stop the program

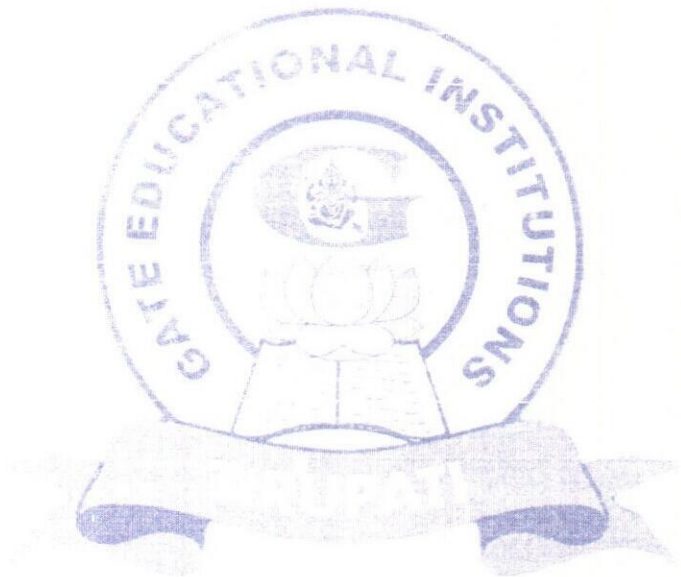
7. Program to implement logic gates in Java.

```
import java.util.Scanner;

public class LogicGates {

  // Logic gate methods
  static int AND(int a, int b) {
     return a & b;
  }

  static int OR(int a, int b) {
     return a | b;
  }

  static int NOT(int a) {
     return (a == 0) ? 1 : 0;
  }

  static int XOR(int a, int b) {
     return a ^ b;
  }

  static int NAND(int a, int b) {
     return (a & b) == 1 ? 0 : 1;
  }

  static int NOR(int a, int b) {
     return (a | b) == 1 ? 0 : 1;
  }
```

```
static int XNOR(int a, int b) {
    return (a ^ b) == 1 ? 0 : 1;
}

public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    int choice, a = 0, b = 0;

    System.out.println("Logic Gates Implementation");
    System.out.println("==========");
    System.out.println("1. AND");
    System.out.println("2. OR");
    System.out.println("3. NOT");
    System.out.println("4. XOR");
    System.out.println("5. NAND");
    System.out.println("6. NOR");
    System.out.println("7. XNOR");

    System.out.print("\nEnter your choice (1-7): ");
    choice = sc.nextInt();

    if (choice == 3) {
        System.out.print("Enter input (0 or 1): ");
        a = sc.nextInt();
    } else {
        System.out.print("Enter first input (0 or 1): ");
        a = sc.nextInt();
```

```
        System.out.print("Enter second input (0 or 1): ");
        b = sc.nextInt();
    }

    // Input validation
    if ((a != 0 && a != 1) || (choice != 3 && (b != 0 && b != 1))) {
        System.out.println("Invalid input! Inputs must be 0 or 1.");
        sc.close();
        return;
    }

    switch (choice) {
        case 1:
            System.out.println("AND(" + a + ", " + b + ") = " + AND(a, b));
            break;
        case 2:
            System.out.println("OR(" + a + ", " + b + ") = " + OR(a, b));
            break;
        case 3:
            System.out.println("NOT(" + a + ") = " + NOT(a));
            break;
        case 4:
            System.out.println("XOR(" + a + ", " + b + ") = " + XOR(a, b));
            break;
        case 5:
            System.out.println("NAND(" + a + ", " + b + ") = " + NAND(a, b));
            break;
        case 6:
            System.out.println("NOR(" + a + ", " + b + ") = " + NOR(a, b));
```

```
            break;
        case 7:
            System.out.println("XNOR(" + a + ", " + b + ") = " + XNOR(a, b));
            break;
        default:
            System.out.println("Invalid choice!");
    }

    sc.close();
  }
}
```



OUTPUT:

Logic Gates Implementation

==========

1. AND
2. OR
3. NOT
4. XOR
5. NAND
6. NOR
7. XNOR

Enter your choice (1-7):1

Enter first input (0 or 1): 1

Enter second input (0 or 1): 0

AND(1, 0) = 0

Enter your choice (1-7):2

Enter first input (0 or 1): 1

Enter second input (0 or 1): 0

OR(1, 0) = 1

Enter your choice (1-7):3

Enter input (0 or 1): 1

NOT(1) = 0

Enter your choice (1-7):4

Enter first input (0 or 1): 1

Enter second input (0 or 1): 0

XOR(1, 0) = 1

Enter your choice (1-7):5

Enter first input (0 or 1): 1

Enter second input (0 or 1): 1

NAND(1, 1) = 0

Enter your choice (1-7):6

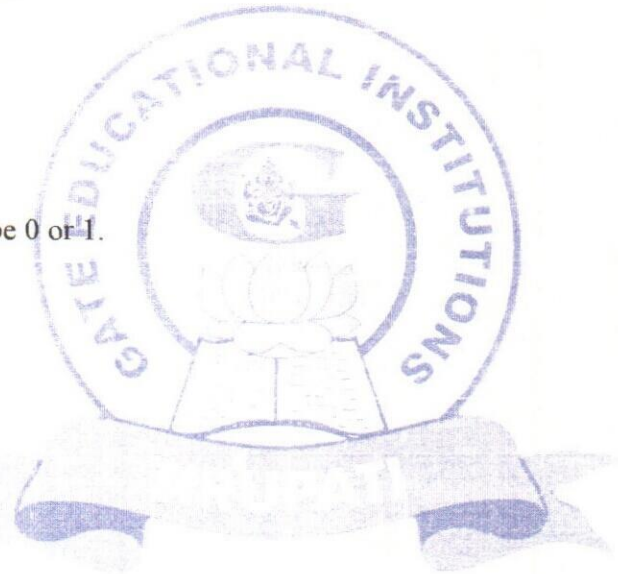
Enter first input (0 or 1): 1

Enter second input (0 or 1): 1

XNOR(1, 1) = 1

Enter your choice (1-7):6

Invalid input! Inputs must be 0 or 1.



8. Write a Java Program to implement Half adder and full adder

Aim:- To implement Half adder and full adder using Java

Description:-

This Program implements binary addition circuits: Half Adder and Full Adder using the Java Programming language

The Program accepts binary inputs (0 or 1) from the user and calculates the Sum and carry using bitwise operators

* Half Adder adds two single-bit binary numbers
* Full Adder adds three single-bit binary numbers (including carry-in)

A menu-driven approach allows the user to choose between Half Adder and Full Adder

Algorithm:-

Step 1: Start the Program

Step 2: Display menu

1. Half Adder
2. Full Adder

Step 3: Read the user's choice

Step 4: Read input values

* If Half Adder is selected:
  * Read two inputs a and b
* If Full Adder is selected:
  * Read three inputs a, b and c in

Step 5: Validate inputs
* check whether all inputs are either 0 or 1
* If invalid, display error message and stop

Step 6:- Perform selected operation

Half Adder
1. compute sum using a XOR b
2. compute carry using a AND b
3. Display sum and carry

Full Adder
1. compute sum using a XOR b XOR c in
2. compute carry using
   (a AND b) OR (b AND cin) OR (a AND cin)
3. Display sum and carry

step 7: Stop the program

8. **Java Program: Half Adder and Full Adder.**

```
import java.util.Scanner;

public class BinaryAdders {

  // Half Adder
  static void halfAdder(int a, int b) {
     int sum = a ^ b;
     int carry = a & b;

     System.out.println("Half Adder Result:");
     System.out.println("Sum = " + sum);
     System.out.println("Carry = " + carry);
  }

  // Full Adder
  static void fullAdder(int a, int b, int cin) {
     int sum = a ^ b ^ cin;
     int carry = (a & b) | (b & cin) | (a & cin);

     System.out.println("Full Adder Result:");
     System.out.println("Sum = " + sum);
     System.out.println("Carry = " + carry);
  }

  public static void main(String[] args) {

     Scanner sc = new Scanner(System.in);
```

```
int choice;
int a, b, cin;

System.out.println("Binary Adder Menu");
System.out.println("==================");
System.out.println("1. Half Adder");
System.out.println("2. Full Adder");

System.out.print("Enter your choice (1 or 2): ");
choice = sc.nextInt();

if (choice == 1) {
   System.out.print("Enter first input (0 or 1): ");
   a = sc.nextInt();
   System.out.print("Enter second input (0 or 1): ");
   b = sc.nextInt();

   if ((a != 0 && a != 1) || (b != 0 && b != 1)) {
      System.out.println("Invalid input! Inputs must be 0 or 1.");
      sc.close();
      return;
   }

   halfAdder(a, b);

} else if (choice == 2) {
   System.out.print("Enter first input (0 or 1): ");
   a = sc.nextInt();
   System.out.print("Enter second input (0 or 1): ");
```

```
        b = sc.nextInt();
        System.out.print("Enter carry-in input (0 or 1): ");
        cin = sc.nextInt();

        if ((a != 0 && a != 1) || (b != 0 && b != 1) || (cin != 0 && cin != 1)) {
            System.out.println("Invalid input! Inputs must be 0 or 1.");
            sc.close();
            return;
        }

        fullAdder(a, b, cin);

    } else {
        System.out.println("Invalid choice! Please select 1 or 2.");
    }

    sc.close();
  }
}
```

OUTPUT:

Binary Adder Menu

==========

1. Half Adder

2. Full Adder

Enter your choice (1 or 2): 2

Enter first input (0 or 1): 1

Enter second input (0 or 1): 1

Enter carry-in input (0 or 1): 1

Full Adder Result:

Sum = 1

Carry = 1