

**EXPERIMENT 1:****WRITE A 8051 C PROGRAM TO MULTIPLY TWO 16 BIT NUMBERS****AIM:**

To multiply two 16 bit numbers using a C program.

**SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

**PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

**PROGRAM:**

```
#include <reg51.h>

void main() {
    while(1) {
        unsigned int num1,num2;
        unsigned long int product;
        num1 = 3;
        num2 = 5;
        product = (unsigned long int ) num1*num2;
    }
}
```

## OUTPUT:

The screenshot shows the Keil MDK-ARM IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations and debugging. The main window is divided into several panes:

- Registers** pane: Shows the state of registers R0 through R7 and system registers like sp\_max, dptr, PC, states, sec, and pw.
- Disassembly** pane: Displays assembly code for the main function, including instructions like SJMP and MOV.
- CODE** pane: Shows the C source code for the main() function, which multiplies two 16-bit numbers (3 and 5) and stores the result (15) in a variable.
- Command** pane: Displays build logs and command-line options.
- Call Stack + Locals** pane: Shows the call stack and the current local variables: num1 (3), num2 (5), and product (15).

The C code in the CODE pane:

```
#include <reg51.h>
void main() {
    unsigned int num1,num2;
    unsigned long int product;
    num1 = 3;
    num2 = 5;
    product = (unsigned long int ) num1*num2;
}
```

## RESULT:

The multiplication of two 16 bit numbers using the 8051 C program is successful.

## **EXPERIMENT 2:**

### **WRITE A 8051 C PROGRAM TO FIND THE SUM OF FIRST 10 INTEGER NUMBERS**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

#### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main(){
    unsigned char sum=0;
    unsigned char i;
    for(i=1;i<=10;i++){
        sum = sum + i;
    }
    ACC = sum;
    P0 = sum;
}
```

## OUTPUT:

The screenshot shows the Keil MDK-ARM IDE interface with the following windows open:

- Registers**: Shows the state of various registers (r0-r7, Sys, sp\_max, dptr, PC \$, states, sec, bsw) with their corresponding memory addresses and values.
- Disassembly**: Displays the assembly code for the program. The current instruction is highlighted at address 0x00000004, which is a MOV R6, R6 instruction. The next instruction at address 0x00000005 is an ADD R7, R6, R7 instruction.
- Code Editor**: Shows the C source code for the program. It includes an include directive, a main function declaration, and a loop that adds integers from 1 to 10 to a variable sum.
- Call Stack + Locals**: Shows the call stack and local variables. The stack frame for the MAIN function is active, containing variables sum (45) and i (10), both of type uchar.
- Command Line**: Displays the command-line interface with the following text:

```
Running with Code Size Limit: 2K
Load "C:\\Raffiq\\Embeddeb c\\EX_2\\Objects\\prog_2"
```
- Status Bar**: Shows simulation information: t1: 0.00022300 sec, L7 C1, CAP NUM SCRL OVR R/M.

## RESULT:

The sum of first 10 integer numbers using 8051 C program is successful

## **EXPERIMENT 3:**

# **WRITE A 8051 C PROGRAM TO FIND FACTORIAL OF A GIVEN NUMBER**

### **AIM:**

To multiply two 16 bit numbers using a C program.

### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software.

### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

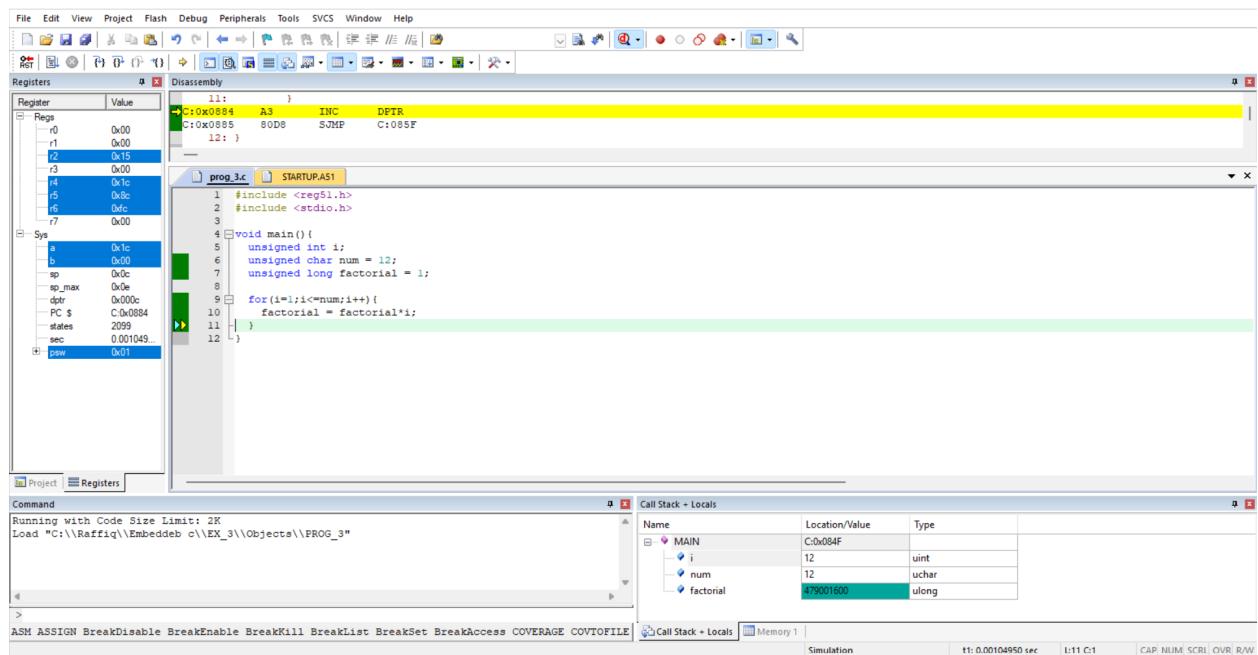
### **PROGRAM:**

```
#include <reg51.h>
#include <stdio.h>

void main(){
    unsigned int i;
    unsigned char num = 12;
    unsigned long factorial = 1;

    for(i=1;i<=num;i++){
        factorial = factorial*i;
    }
}
```

## OUTPUT:



## RESULT:

The verification of factorial of given number is successful using 8051 C program

## **EXPERIMENT 4:**

### **WRITE A 8051 C PROGRAM TO ADD AN ARRAY OF 16-BIT NUMBERS AND STORE THE 32-BIT RESULT IN INTERNAL RAM**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

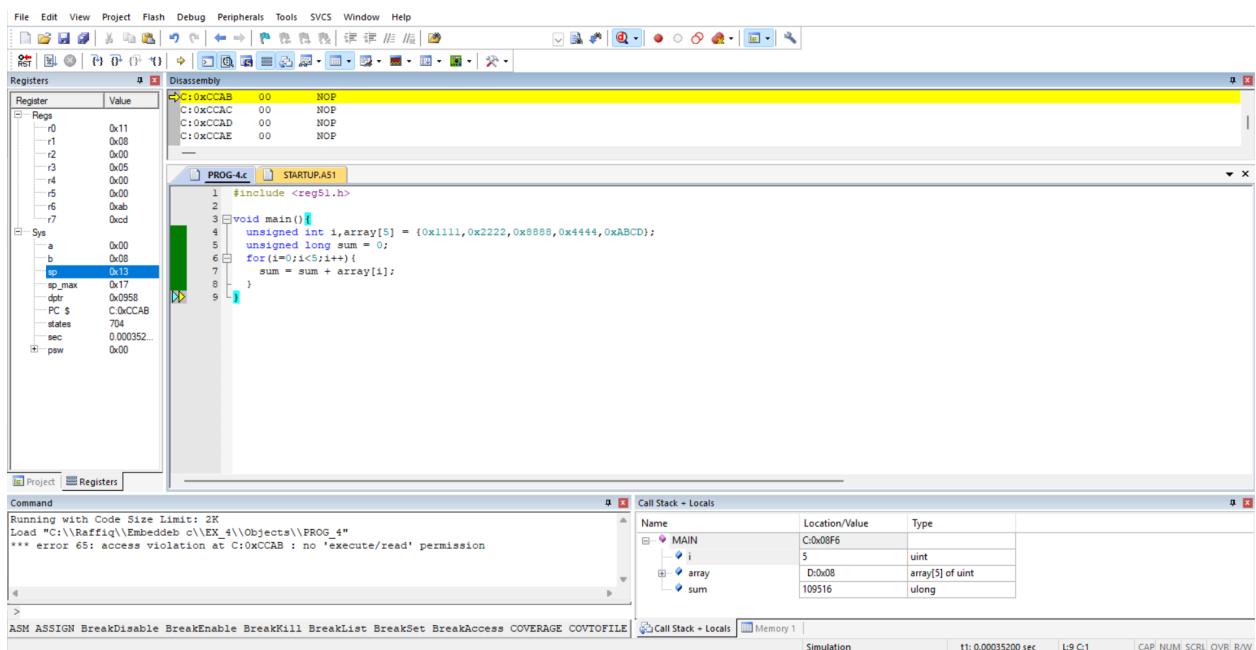
- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main(){
    unsigned int i,array[5] = {0x1111,0x2222,0x8888,0x4444,0xABCD};
    unsigned long sum = 0;
    for(i=0;i<5;i++){
        sum = sum + array[i];
    }
}
```

## OUTPUT:



## RESULT:

The verification of 32-bit sum of the given 16-bit numbers is stored in internal RAM starting from address 0x30 is successful using C program

## **EXPERIMENT 5:**

### **WRITE A 8051 C PROGRAM TO FIND THE SQUARE OF A NUMBER (1 TO 10) USING LOOK-UP TABLE**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

#### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main(){
    unsigned char LUT[] = {1,4,9,16,25,36,49,64,81,100};
    unsigned char num,square;
    for(num=1;num<11;num++){
        square = LUT[num-1];
        P0 = square;
    }
}
```

## OUTPUT:

The screenshot shows a debugger interface with several windows:

- Registers**: Shows CPU registers (R0-R7, SP, SP\_max) with their current values.
- Disassembly**: Shows assembly code for the main function, including instructions like MOV, ADD, and MOVW.
- Project**: Shows the project files: prog\_5.c and STARTUP.A51.
- Call Stack + Locals**: Shows the call stack and local variable table for the MAIN function. The LUT array contains values 1 through 100, corresponding to the square of numbers 1 through 10.
- Command**: Displays the command line with the message "Running with Code Size Limit: 2K".

Name	Location/Value	Type
MAIN	C:0x08F6	
LUT	D:0x08`000000\$1@...	array[10] of uchar
[0]	1	uchar
[1]	4	uchar
[2]	9	uchar
[3]	16	uchar
[4]	25	uchar
[5]	36	uchar
[6]	49	uchar
[7]	64	uchar
[8]	81	uchar
[9]	100	uchar
num	10	uchar
square	100	uchar

## RESULT:

The verification of square of number(1 to 10) using look up table is successful using C program

## **EXPERIMENT 6:**

### **WRITE A 8051 C PROGRAM TO FIND THE LARGEST/SMALLEST NUMBER IN AN ARRAY OF NUMBERS**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main(){
    unsigned long array[] = {0x33334444,0x99998888,0xffffbbbb,0x55559999,0x11110000};
    unsigned long i,largest = 0;
    for(i=0;i<5;i++){
        if(largest<array[i])
            largest = array[i];
    }
}
```

## OUTPUT:

The screenshot shows the µVision IDE interface with the following details:

- Registers:** Shows CPU registers (R0-R7, Sys) with their values.
- Disassembly:** Shows assembly code for the `prog_6.c` file. The assembly code includes instructions like `RET`, `RLC`, and the main loop logic.
- Code Editor:** Displays the C source code for `prog_6.c`:
 

```
#include <reg51.h>
void main()
{
    unsigned long array[] = {0x33334444, 0x99998888, 0xffffbbbb, 0x55559999, 0x11110000};
    unsigned long i, largest = 0;
    for(i=0; i<5; i++)
        if(largest<array[i])
            largest = array[i];
}
```
- Call Stack + Locals:** Shows the variable `array` and its elements, along with `i` and `largest`.
- Command Line:** Displays the command used to run the simulation: "Running with Code Size Limit: 2K Load C:\Raffiq\Embeddeb c:\EX\_6\Objects\prog\_6".
- Status Bar:** Shows simulation time (t1: 0.00060750 sec), memory usage (L10 C1), and other system information.

The screenshot shows the µVision IDE interface with the following details:

- Registers:** Shows CPU registers (R0-R7, Sys) with their values.
- Disassembly:** Shows assembly code for the `EXXX.c` file. The assembly code includes instructions like `NOP` and the main loop logic.
- Code Editor:** Displays the C source code for `EXXX.c`:
 

```
#include <reg51.h>
void main()
{
    unsigned long array[] = {0x33334444, 0x10008888, 0xffffbbbb, 0x00009999, 0x11110000};
    unsigned long i, smallest = 0xFFFFFFF;
    for(i = 0; i < 5; i++)
        if (smallest > array[i])
            smallest = array[i];
}
```
- Call Stack + Locals:** Shows the variable `array` and its elements, along with `i` and `smallest`.
- Command Line:** Displays the command used to run the simulation: "Running with Code Size Limit: 2K Load C:\Raffiq\Embeddeb c:\ex10B\Objects\EXXXX".
- Status Bar:** Shows simulation time (t1: 0.00099600 sec), memory usage (L18 C1), and other system information.

## RESULT:

The verification of to find the largest and smallest number in an array of numbers using C program is successful

## **EXPERIMENT 7:**

### **WRITE A 8051 C PROGRAM TO ARRANGE A SERIES OF NUMBERS IN ASCENDING/DESCENDING ORDER LOCATIONS**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void bubble_sort(unsigned char *arr, unsigned char n, bit ascending) {
    unsigned char i, j, temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - 1 - i; j++) {
            if ((ascending && arr[j] > arr[j + 1]) || (!ascending && arr[j] < arr[j + 1])) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

```

    }
}
}

void main() {
    unsigned char data numbers[] = {25, 10, 5, 30, 15};
    unsigned char n = 5, i;
    bit order = 1; // 1 for ascending, 0 for descending

    bubble_sort(numbers, n, order);

    while (1);
}

```

## OUTPUT:

The screenshot shows a debugger interface with several windows:

- Registers**: Shows CPU registers (R0-R7, Sys, SP, PC, etc.) and their values.
- Disassembly**: Shows assembly code with highlighted instructions. The highlighted instruction is `C10x09F3 D200 SETB 0x20.0`.
- Code View**: Shows the C source code for `EXPERIMENT_7.c`, including the `bubble_sort` function and the `main` function call.
- Call Stack + Locals**: Shows the call stack and local variable values. The local variables are:
 

Name	Location/Value	Type
MAIN	C009DE	
numbers	D0x08 "	array[5] of uchar
n	0	uchar
i	0	uchar
order	0	bit
- Command**: Shows the command line interface with the current project path and build status.

The screenshot shows the Keil MDK-ARM IDE interface. The top menu includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations and debugging. The main window has several panes:

- Registers**: Shows the state of registers r0 through r7 and system registers sys.
- Disassembly**: Displays assembly code, including the RET instruction at address 0x00090DD0 and the main() function definition.
- Code Editor (EXPERIMENT\_7.c)**: Shows the C source code for a bubble sort algorithm. The code defines a bubble\_sort function and a main function that calls it with an array of numbers and a sorting order.
- Call Stack + Locals**: A table showing the current stack frame with variables numbers, n, i, and order.
- Command Line**: Displays build logs and command-line options.

```

14: )
15:
C:\0x090DD0    22      RET
16: void main() {
3   void bubble_sort(unsigned char *arr, unsigned char n, bit ascending) {
4     unsigned char i, j, temp;
5     for (i = 0; i < n - 1; i++) {
6       for (j = 0; j < n - 1 - i; j++) {
7         if ((ascending && arr[j] > arr[j + 1]) || (!ascending && arr[j] < arr[j + 1])) {
8           temp = arr[j];
9           arr[j] = arr[j + 1];
10          arr[j + 1] = temp;
11        }
12      }
13    }
14  }
15
16 void main() {
17   unsigned char data_numbers[] = {25, 10, 5, 30, 15};
18   unsigned char n = 5, i;
19   bit order = 1; // 1 for ascending, 0 for descending
20
21   bubble_sort(numbers, n, order);
22
23   while (1);
24 }

```

Name	Location/Value	Type
MAIN	C0x09DE	
numbers	D0x08 "D\nC0"	array[5] of uchar
n	5	uchar
i	0	uchar
order	bit	

## RESULT:

To arrange series of numbers in ascending and descending order using C program is successful

## **EXPERIMENT 8:**

### **WRITE A 8051 C PROGRAM TO COUNT THE NUMBER OF ONES AND ZEROS IN TWO CONSECUTIVE MEMORY LOCATIONS**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main()
{
    unsigned long array[] = {0x33556666, 0xCCAADD00, 0x55998888, 0x77664444,
0x11223344};
    unsigned long temp;
    unsigned int i, j;

    // Bubble Sort Algorithm for Descending Order
    for(i = 0; i < 5 - 1; i++)
```

```

{
    for(j = 0; j < 5 - 1 - i; j++)
    {
        if(array[j] < array[j + 1])
        {
            temp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = temp;
        }
    }
}

while(1); // Infinite loop to keep the program running
}

```

## OUTPUT:

The screenshot shows the Keil MDK-ARM IDE interface with the following windows open:

- Registers**: Shows the initial state of registers (r0-r7, sp, pc, etc.) with values like 0x00.
- Disassembly**: Shows the assembly code for the main() function. The assembly code corresponds to the C code provided above, including the bubble sort logic.
- Call Stack + Locals**: Shows the local variables and their initial values:
 

Name	Location/Value	Type
MAIN	C0092C	
array	D0x08	array[5] of ulong
temp	0x00000000	ulong
i	0x0000	uint
j	0x0000	uint
- Command**: Shows the command line interface with the current project path and build status.

## RESULT:

To count the number of ones and Zeros in two consecutive memory locations using C program is successful.

## **EXPERIMENT 9:**

### **WRITE A 8051 C PROGRAM TO SCAN A SERIES OF NUMBERS TO FIND HOW MANY ARE NEGATIVE**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main() {
    signed char numbers[] = {5, -3, 12, -7, 0, -1, 9, -8};
    unsigned char i, count = 0;
    unsigned char length = sizeof(numbers) / sizeof(numbers[0]);

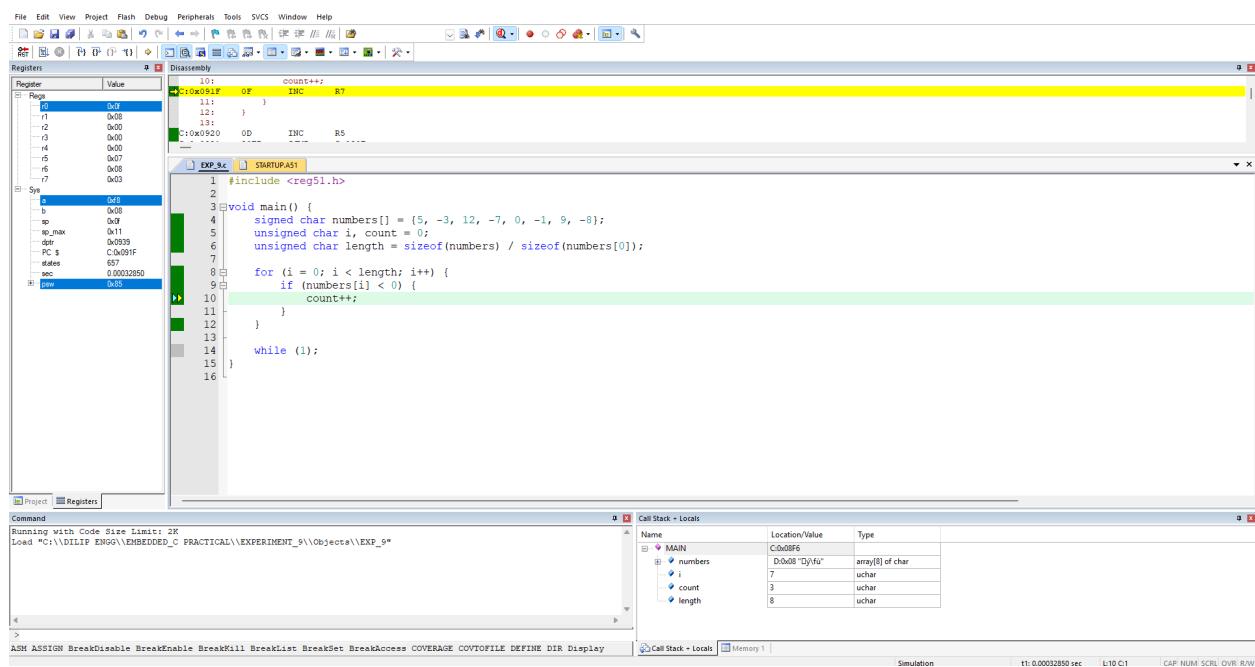
    for (i = 0; i < length; i++) {
        if (numbers[i] < 0) {
            count++;
        }
    }
}
```

```

while (1);
}

```

## OUTPUT:



## RESULT:

To scan a series of numbers to find how many are negative is successful using a C program.

## **EXPERIMENT 10:**

### **WRITE A 8051 C PROGRAM TO DISPLAY "HELLO WORLD" MESSAGE (EITHER IN SIMULATION MODE OR INTERFACE AN LCD DISPLAY)**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

#### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

// Define LCD control pins
sbit RS = P2^0; // Register Select
sbit RW = P2^1; // Read/Write
sbit EN = P2^2; // Enable

// Function prototypes
void delay(unsigned int);
void lcd_command(unsigned char);
```

```
void lcd_data(unsigned char);
void lcd_init();
void lcd_string(char *str);

void main() {
    lcd_init(); // Initialize LCD

    lcd_string("HELLO WORLD"); // Display "HELLO WORLD"

    while (1); // Infinite loop
}

// Function to generate delay
void delay(unsigned int time) {
    unsigned int i, j;
    for (i = 0; i < time; i++)
        for (j = 0; j < 1275; j++); // Adjust delay as needed
}

// Function to send command to LCD
void lcd_command(unsigned char cmd) {
    P1 = cmd; // Send command to data port
    RS = 0; // Command mode
    RW = 0; // Write mode
    EN = 1;
    delay(2);
    EN = 0;
}

// Function to send data to LCD
void lcd_data(unsigned char dat) { // Corrected parameter name
    P1 = dat; // Send data to data port
    RS = 1; // Data mode
    RW = 0; // Write mode
    EN = 1;
    delay(2);
    EN = 0;
}

// Function to initialize LCD
```

```

void lcd_init() {
    lcd_command(0x38); // 2-line, 5x7 matrix
    delay(5);
    lcd_command(0x0C); // Display ON, Cursor OFF
    delay(5);
    lcd_command(0x06); // Increment cursor
    delay(5);
    lcd_command(0x01); // Clear display
    delay(10);
}

```

```

// Function to send a string to LCD
void lcd_string(char *str) {
    while (*str) {
        lcd_data(*str);
        str++;
    }
}

```

## OUTPUT:

The screenshot shows a debugger interface with several windows:

- File Edit View Project Flash Debug Peripherals Tools SVCS Window Help**: The menu bar.
- Registers**: Shows CPU registers (r0-r7, a, b, dpc, dptr, states, mem, psw) all set to 0x00.
- Disassembly**: Shows the assembly code for the program. The main loop starts at address C006A1 and calls the lcd\_init function.
- Parallel Port 1**: Shows the port configuration for Port 1.
- Project**: Shows the project files.
- Command**: Shows the command line options used to run the program.
- Call Stack - Locals**: Shows the call stack and local variables, with MAIN at C006A1.
- Memory**: Shows memory dump and simulation status.

## RESULT:

The display of Message in port is successful using the C program.

## **EXPERIMENT 11:**

### **WRITE A 8051 C PROGRAM TO CONVERT THE HEXADECIMAL DATA 0XCFH TO DECIMAL AND DISPLAY THE DIGITS ON PORTS P0, P1, AND P2 (PORT WINDOW IN SIMULATOR)**

#### **AIM:**

To multiply two 16 bit numbers using a C program.

#### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

#### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char hex_value = 0xCF; // Hexadecimal value (207 in decimal)
    unsigned char hundreds, tens, ones;
    unsigned char decimal_value;

    // Convert hex to decimal
    decimal_value = hex_value; // 0xCF = 207 (Decimal)
```

```

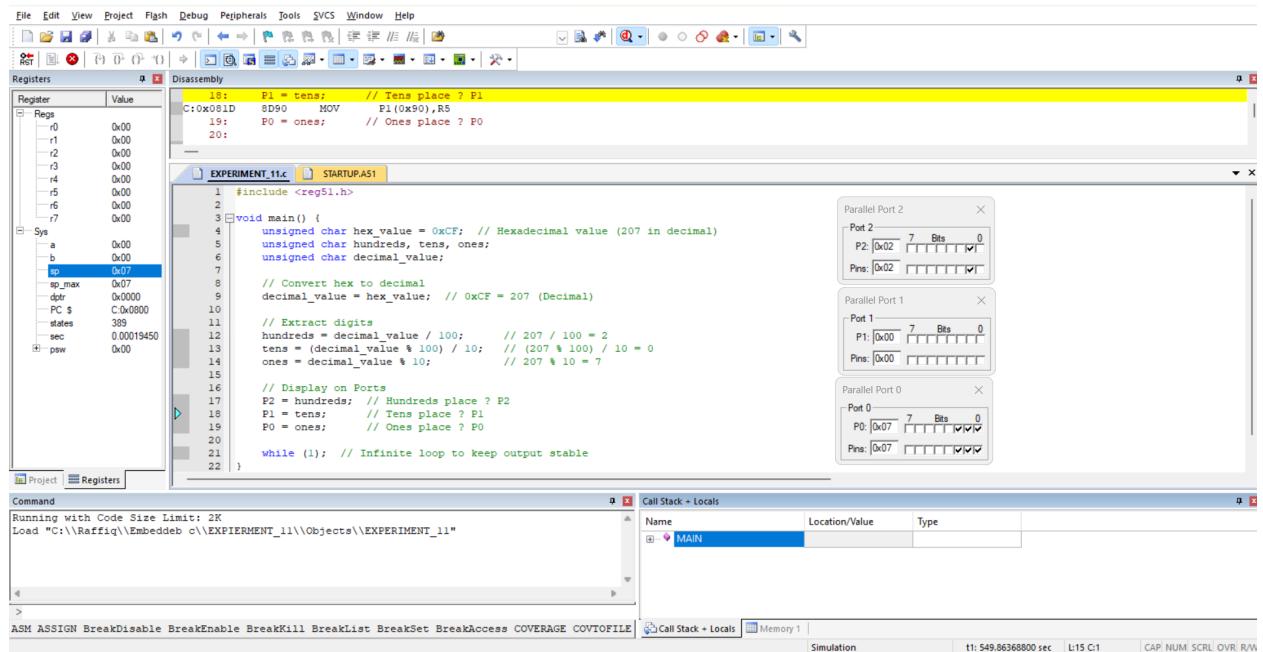
// Extract digits
hundreds = decimal_value / 100; // 207 / 100 = 2
tens = (decimal_value % 100) / 10; // (207 % 100) / 10 = 0
ones = decimal_value % 10; // 207 % 10 = 7

// Display on Ports
P2 = hundreds; // Hundreds place ? P2
P1 = tens; // Tens place ? P1
P0 = ones; // Ones place ? P0

while (1); // Infinite loop to keep output stable
}

```

## OUTPUT:



## RESULT:

The program to convert an hexadecimal data to decimal data and display in port 0,1,2 is successful using C program

## **EXPERIMENT 12:**

**Write a 8051 C program to generate and print the first 10 numbers in the Fibonacci sequence**

### **AIM:**

To generate and print the first 10 numbers in the fibonacci sequence

### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned int fib[10];
    unsigned char i;

    fib[0] = 0;
    fib[1] = 1;

    for (i = 2; i < 10; i++) {
        fib[i] = fib[i - 1] + fib[i - 2];
    }
}
```

}

```
while (1);  
}
```

## OUTPUT:

The screenshot shows a debugger interface with several windows:

- Registers**: Shows CPU registers (R0-R7, SP, PC, etc.) with their current values.
- Disassembly**: Shows assembly code for the current program. The highlighted line is `C10x0835 80FE SJMP C10835`.
- Source View**: Shows the C source code for the Fibonacci sequence generation. The highlighted line is `while (1);`.
- Call Stack + Locals**: Shows the call stack and local variable values. The `fib` array contains values 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 at addresses D0080 to D009F. The variable `i` has a value of 10 at address D00A0.
- Command Line**: Displays the command line with the current project path and build status.

## RESULT:

To generate and print the first 10 numbers in the fibonacci sequence is successful.

## **EXPERIMENT 13:**

**Write a 8051 C program to perform matrix addition of two  $2 \times 2$  matrices**

### **AIM:**

To generate the C program to perform matrix addition of two  $2 \times 2$  matrices.

### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char A[2][2] = {{1, 2}, {3, 4}};
    unsigned char B[2][2] = {{5, 6}, {7, 8}};
    unsigned char C[2][2];
    unsigned char i, j;

    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}
```

```
    }  
  while (1);  
}
```

## OUTPUT:

The screenshot shows a debugger interface with the following components:

- Registers**: Shows CPU registers R0-R7 and SREG with their current values.
- Disassembly**: Displays assembly code for the EXP\_13.c program, highlighting the main() function. The assembly code includes instructions like SWI, MOV, CLR, and ADD.
- Call Stack + Locals**: Shows the call stack and local variable table. The stack contains arrays A, B, C, and integer variables i and j. The arrays are 2x2 uchar arrays, and i/j are uchar variables.

## RESULT:

The generation of the C program to perform matrix addition of two 2x2 matrices is successful

## **EXPERIMENT 14:**

### **Write a C program to check if a given string is a palindrome**

#### **AIM:**

To perform the palindrome operation in a C program using the given string.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main() {
    char str[] = "madam";
    unsigned char i, len = 0, flag = 1;

    while (str[len] != '\0') {
        len++;
    }

    for (i = 0; i < len / 2; i++) {
        if (str[i] != str[len - 1 - i]) {
            flag = 0;
        }
    }
}
```

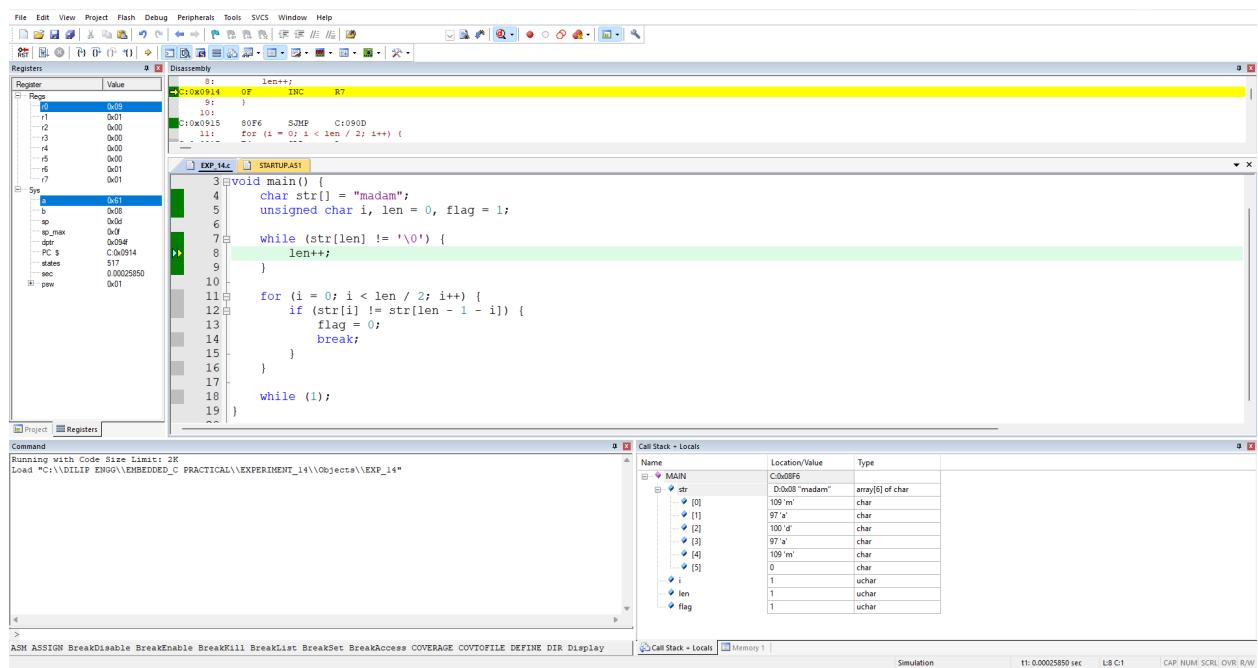
```

        break;
    }
}

while (1);
}

```

## OUTPUT:



## RESULT:

To Perform the operation to check the string is palindrome is successful using C program

## **EXPERIMENT 15:**

**Write a 8051 C program to calculate the greatest common divisor (GCD) of two integers**

### **AIM:**

To calculate the GCD of two integers using C program

### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

unsigned int gcd(unsigned int a, unsigned int b) {
    while (b != 0) {
        unsigned int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

void main() {
```

```

unsigned int num1 = 56, num2 = 98, result;
result = gcd(num1, num2);

while (1);
}

```

## OUTPUT:

The screenshot shows the Keil MDK-ARM IDE interface with several windows open:

- File Edit View Project Flash Debug Peripherals Tools SVCS Window Help**: The main menu bar.
- Registers**: Shows the state of various registers (R0-R7, SP, PC, etc.) with values mostly at 0x00.
- Disassembly**: Shows the assembly code generated from the C code. The assembly code includes instructions like `MOV R0, #IDATALEN - 1` and a loop structure.
- EXP15c [STARTUPA51]**: The C code editor window containing the provided C program for calculating the GCD.
- Call Stack + Locals**: A table showing the current state of variables:

Name	Location/Value	Type
MAIN	C00B97	
num1	14	uint
num2	0	uint
result	14	uint
- Command**: A terminal-like window showing build logs and commands.

## RESULT:

The calculation of GCD of two numbers is successful using C program

## **EXPERIMENT 16:**

**Write a 8051 C program to merge two arrays of integers into a single array**

### **AIM:**

To merge two arrays of integers into a single array using C program.

### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char A[] = {1, 3, 5, 7};
    unsigned char B[] = {2, 4, 6, 8};
    unsigned char C[8];
    unsigned char i, j, k = 0;

    for (i = 0; i < 4; i++) {
        C[k++] = A[i];
    }

    for (j = 0; j < 4; j++) {
```

```

C[k++] = B[j];
}

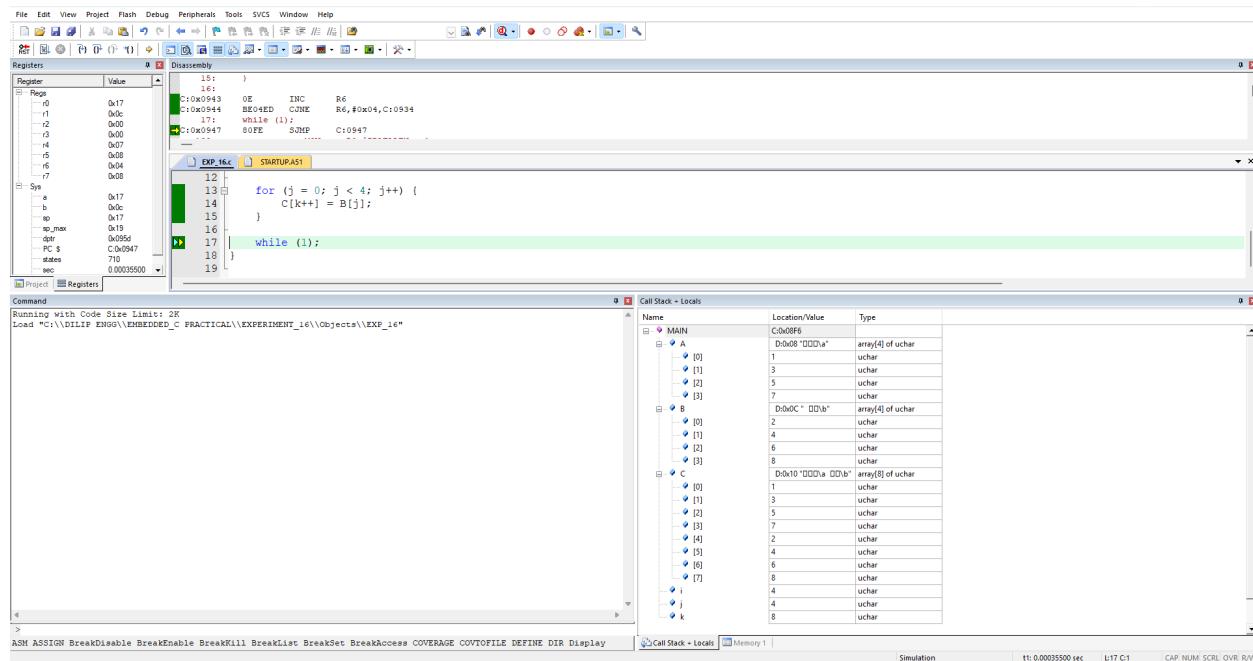
```

```

while (1);
}

```

## OUTPUT:



## RESULT:

The merger of two arrays into a single array is successful using C program

## **EXPERIMENT 17:**

### **Write a 8051 C program to sort an array of integers in ascending order**

#### **AIM:**

To sort an array of integers in the ascending order using the C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char arr[] = {5, 2, 9, 1, 6};
    unsigned char i, j, temp;
    unsigned char n = sizeof(arr) / sizeof(arr[0]);

    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - 1 - i; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

```

    }
}

}

while (1);
}

```

## OUTPUT:

The screenshot shows a debugger interface with the following windows:

- Registers:** Shows CPU registers (R00-R07) with their addresses and values.
- Call Stack - Locals:** Shows the call stack and local variable table for the MAIN function. The table includes:
 

Name	Location/Value	Type
MAIN	C000F6	
arr	D0x08 "0 CD4"	array[5] of uchar
i	1	uchar
j	2	uchar
temp	5	uchar
n	6	uchar
i	9	uchar
j	4	uchar
temp	1	uchar
n	5	uchar
- Memory:** Shows memory dump at address C000F6.
- Assembly:** Shows the assembly code corresponding to the C program.
- C Source Code:** Shows the C code for the main() function.

## RESULT:

The sorting of array in ascending order is successful using a C program.

## **EXPERIMENT 18:**

**Write a 8051 C program to initialize UART communication at 9600 baud rate and send "Hello, World!" (simulation mode)**

### **AIM:**

To initialize UART communication at 9600 baud rate and send "Hello world" using the C program.

### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void uart_init() {
    TMOD = 0x20; // Timer1 mode 2 (8-bit auto-reload)
    TH1 = 0xFD; // Baud rate 9600 at 11.0592MHz
    SCON = 0x50; // 8-bit UART mode, REN enabled
    TR1 = 1; // Start Timer1
}

void uart_tx(char c) {
    SBUF = c;
```

```

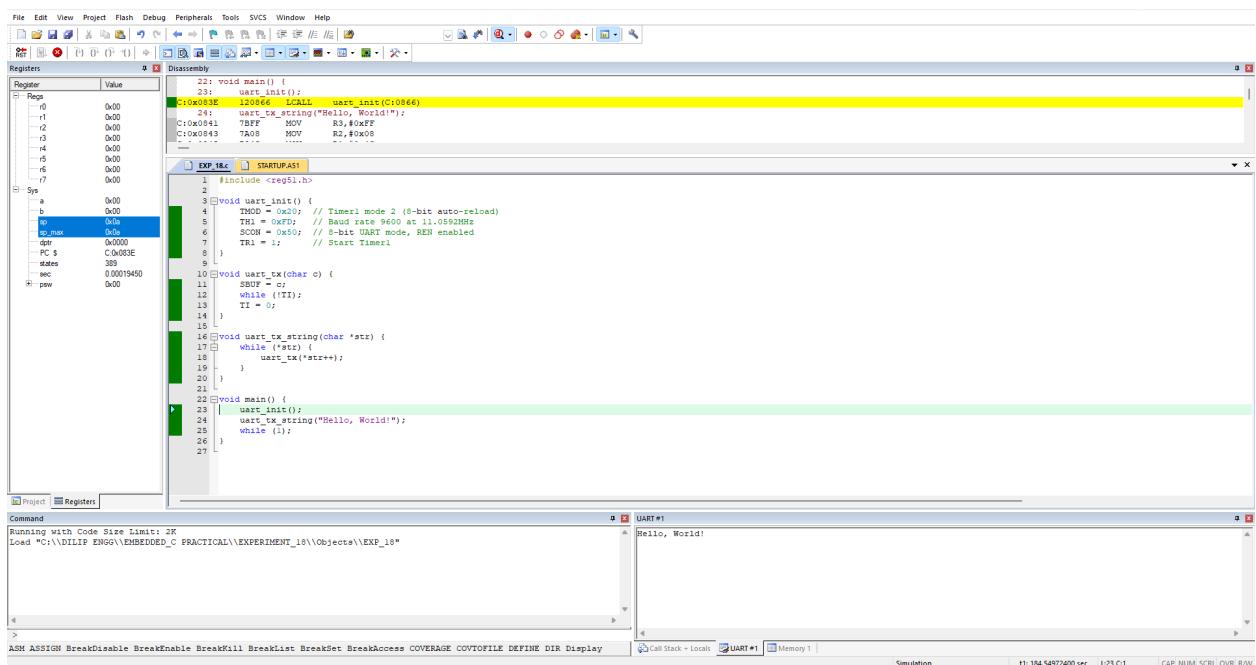
while (!TI);
    TI = 0;
}

void uart_tx_string(char *str) {
    while (*str) {
        uart_tx(*str++);
    }
}

void main() {
    uart_init();
    uart_tx_string("Hello, World!");
    while (1);
}

```

## OUTPUT:



## RESULT:

To initialize UART communication at 9600 baud rate and send "Hello world" using the C program is successful using a C program.

## **EXPERIMENT 19:**

### **Write a C program to find the transpose of a matrix**

#### **AIM:**

To find the transpose of a matrix using a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char A[2][2] = {{1, 2}, {3, 4}};
    unsigned char T[2][2];
    unsigned char i, j;

    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            T[j][i] = A[i][j];
        }
    }
}
```

```

while (1);
}

```

## OUTPUT:

The screenshot shows a debugger interface with the following components:

- Registers:** Shows CPU register values. Notable values include R0 = 0x0F, R1 = 0x08, R2 = 0x00, R3 = 0x00, R4 = 0x00, R5 = 0x04, R6 = 0x02, and R7 = 0x02.
- Datasheet:** Shows assembly code for the current instruction. The assembly code is:
 

```

        14:    while (1);
        15:    00FE    SJMP   C:0927
        16:    C:0927  80FF    MOV    R0,#IDATALEN - 1
        17:    C:0929  787F    MOV    R0,#0x7F
        18:    C:092A  E4      CLR    A
      
```
- C Source Code:** Displays the C code being debugged:
 

```

#include <reg51.h>
void main() {
    unsigned char A[3][3] = {{1, 2}, {3, 4}};
    unsigned char T[3][3];
    unsigned char i, j;
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            T[i][j] = A[j][i];
        }
    }
    while (1);
}
      
```
- Call Stack - Locals:** Shows the call stack and local variable values. The stack frame for MAIN contains:
 

Name	Location/Value	Type
MAIN	C:0895	array[2][2] of uchar
A	0x0000	array[2][2] of uchar
i	0x00 "0"	uchar
j	0x01 "1"	uchar

## RESULT:

To find the transpose of a matrix is successful using a C program.

## **EXPERIMENT 20:**

### **Write a C program to count the frequency of each character in a given string**

#### **AIM:**

To count the frequency of each character in a given string using a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

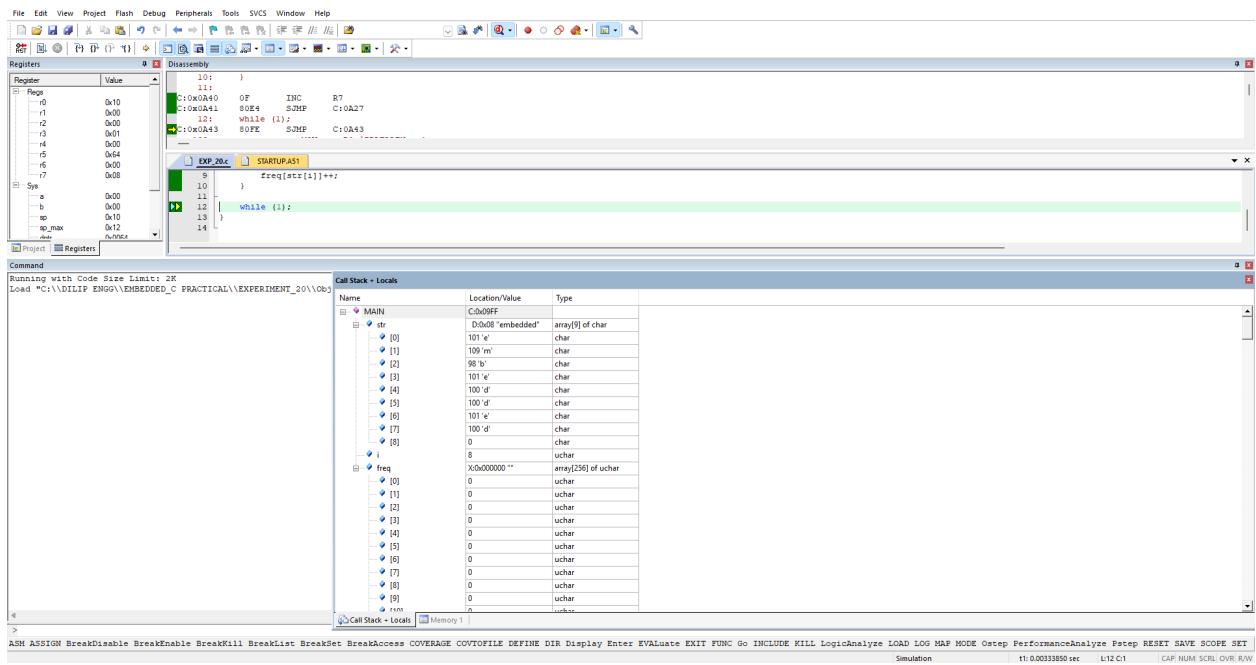
```
#include <reg51.h>

void main() {
    char str[] = "embedded";
    unsigned char i;
    unsigned char xdata freq[256] = {0}; // Use external RAM (xdata)

    for (i = 0; str[i] != '\0'; i++) {
        freq[str[i]]++;
    }

    while (1);
}
```

## OUTPUT:



## RESULT:

To count the frequency of each character in a given string is successful using a C program.

## **EXPERIMENT 21:**

### **Write a C program to find the factorial of a number**

#### **AIM:**

To find the factorial of a number using a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

volatile unsigned char num = 5; // Prevents compiler optimization

unsigned int factorial(unsigned char n) {
    unsigned int fact = 1;
    unsigned char i;
    for (i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}
```

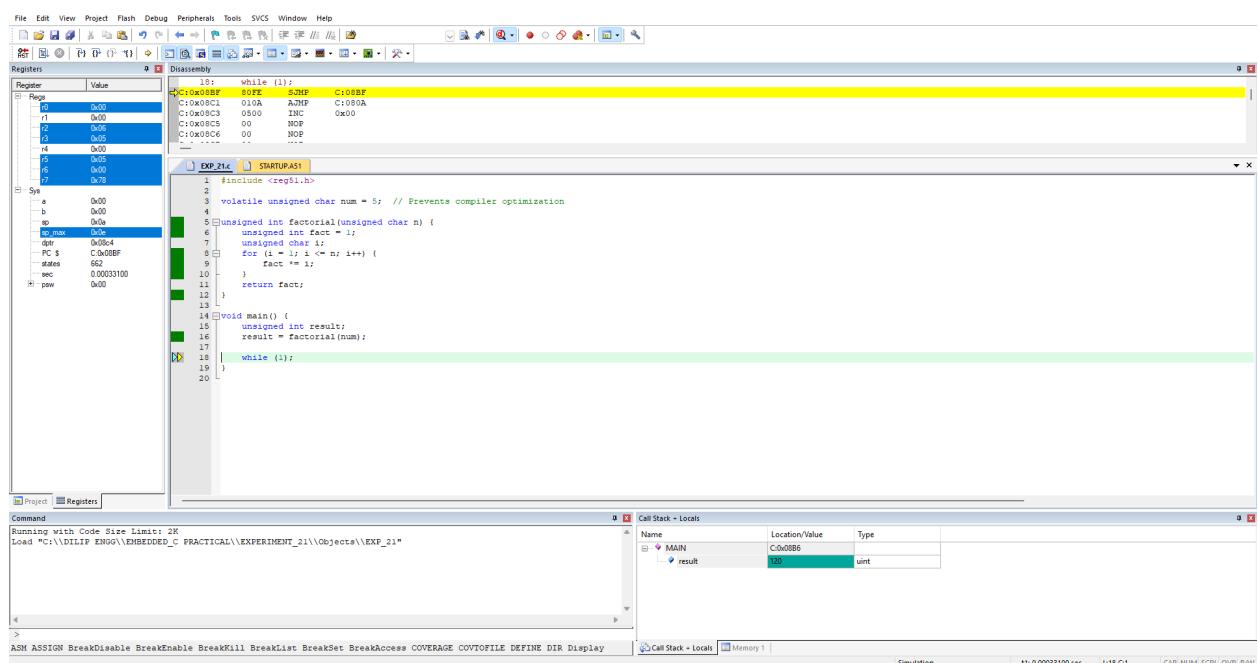
```

void main() {
    unsigned int result;
    result = factorial(num);

    while (1);
}

```

## OUTPUT:



## RESULT:

To find the factorial of a number is successful using a C program.

## **EXPERIMENT 22:**

**Write a 8051 C program to generate the multiplication table for a number (1 to 10) and store it in internal RAM**

### **AIM:**

To generate the multiplication table for a number from (1 to 10) using a C program.

### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char num = 5; // Change this number as needed
    unsigned char i;
    unsigned char table[10]; // Store multiplication table in internal RAM

    for (i = 0; i < 10; i++) {
        table[i] = num * (i + 1); // Store results in table array
    }

    while (1);
```

}

## OUTPUT:

The screenshot shows a debugger interface with the following windows:

- Registers**: Shows CPU register values. R0 to R5 are set to 0x11, R6 to 0x00, and R7 to 0x05.
- Call Stack + Locals**: Shows the call stack and variable table. The variable table contains:

Name	Location/Value	Type
MAIN	C00000	
num	5	uchar
i	10	uchar
table	D008'DnDD#(2)	array[10] of uchar
[0]	5	uchar
[1]	10	uchar
[2]	15	uchar
[3]	20	uchar
[4]	25	uchar
[5]	30	uchar
[6]	35 'W	uchar
[7]	40 'C	uchar
[8]	45 'L	uchar
[9]	50 '2	uchar
- Memory**: Shows memory dump from C00000 to C0001F.
- Project**: Shows the project files.
- Command**: Shows the command line with ASH ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakAccess COVERAGE COVTOFILE DEFINE DIR Display.

## RESULT:

To generate the multiplication table for a number from (1 to 10) is successful using a C program.

## **EXPERIMENT 23:**

**Write a 8051 C program to find and store all prime numbers between 1 and 50 in an array**

### **AIM:**

To find and store all prime numbers between 1 and 50 in an array using a C program.

### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char primes[50];
    unsigned char i, j, count = 0;
    bit isPrime;

    for (i = 2; i <= 50; i++) {
        isPrime = 1;
        for (j = 2; j * j <= i; j++) {
            if (i % j == 0) {
                isPrime = 0;
            }
        }
        if (isPrime == 1) {
            primes[count] = i;
            count++;
        }
    }
}
```

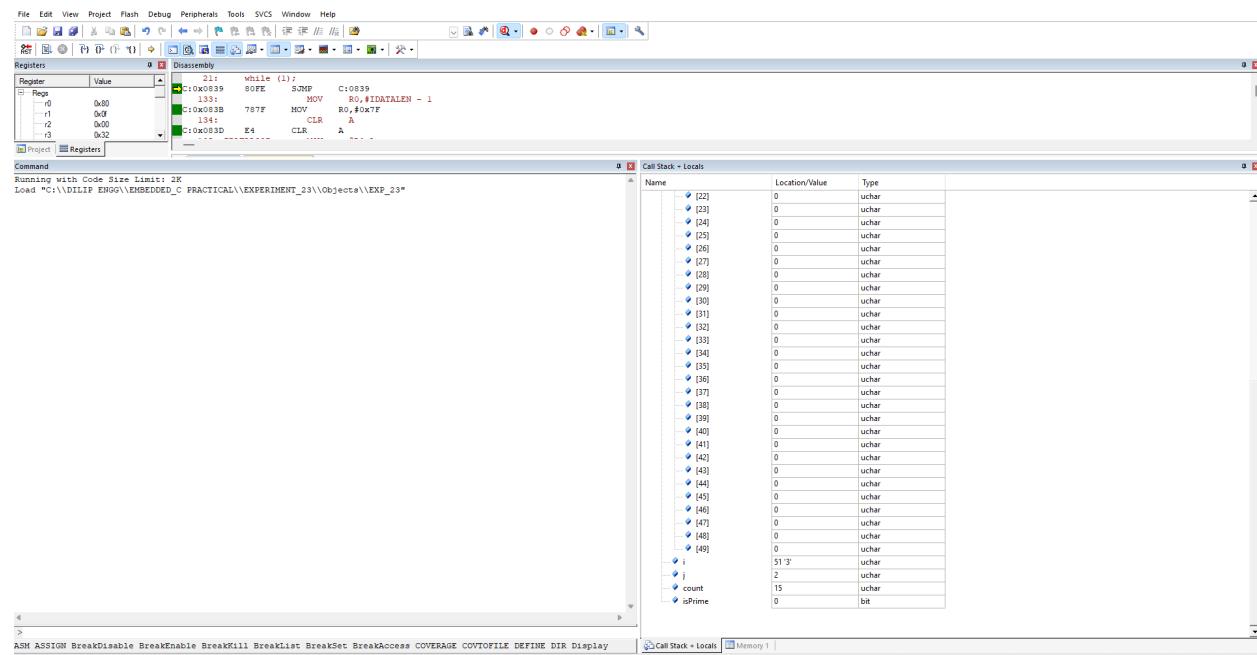
```

        break;
    }
}
if (isPrime) {
    primes[count++] = i;
}
}

while (1);
}

```

## OUTPUT:



## RESULT:

To find and store all prime numbers between 1 and 50 in an array is successful using a C program.

## **EXPERIMENT 24:**

**Write a 8051 C program to reverse the digits of a 16-bit number and store the result in memory**

### **AIM:**

To reverse the digits of a 16-bit number and store result in memory using a C program.

### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned int num = 1234; // Example 16-bit number
    unsigned int rev = 0;

    while (num > 0) {
        rev = (rev * 10) + (num % 10);
        num /= 10;
    }

    while (1);
```

}

## OUTPUT:

The screenshot shows a debugger interface with the following windows:

- Registers**: Shows CPU registers R0-R7 and SREG with their current values.
- Disassembly**: Shows assembly code for the program, including instructions like SETB, MOV, SUB, and JZ.
- CODE**: Shows the C source code for the program, which reads a 16-bit number, reverses its digits, and stores the result in memory.
- Call Stack + Locals**: Shows the call stack and local variables for the MAIN function, with 'num' set to 1234 and 'rev' set to 4321.
- Command**: Displays the command line with the loaded file path: "Running with Code Size Limit: 2K Load \"C:\\DILIP\\ENGG\\EMBEDDED\_C\\PRACTICAL\\EXPERIMENT\_24\\Objects\\EXP\_24\"".
- Memory**: Shows the memory dump with address 0x00000000 containing the value 1234.

## RESULT:

To reverse the digits of a 16-bit number and store the result in memory is successful using a C program.

## **EXPERIMENT 25:**

### **Write a 8051 C program to check whether a given 3-digit number is an Armstrong number**

#### **AIM:**

To check whether a given 3-digit number is an Armstrong number using a C program.

#### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

#### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

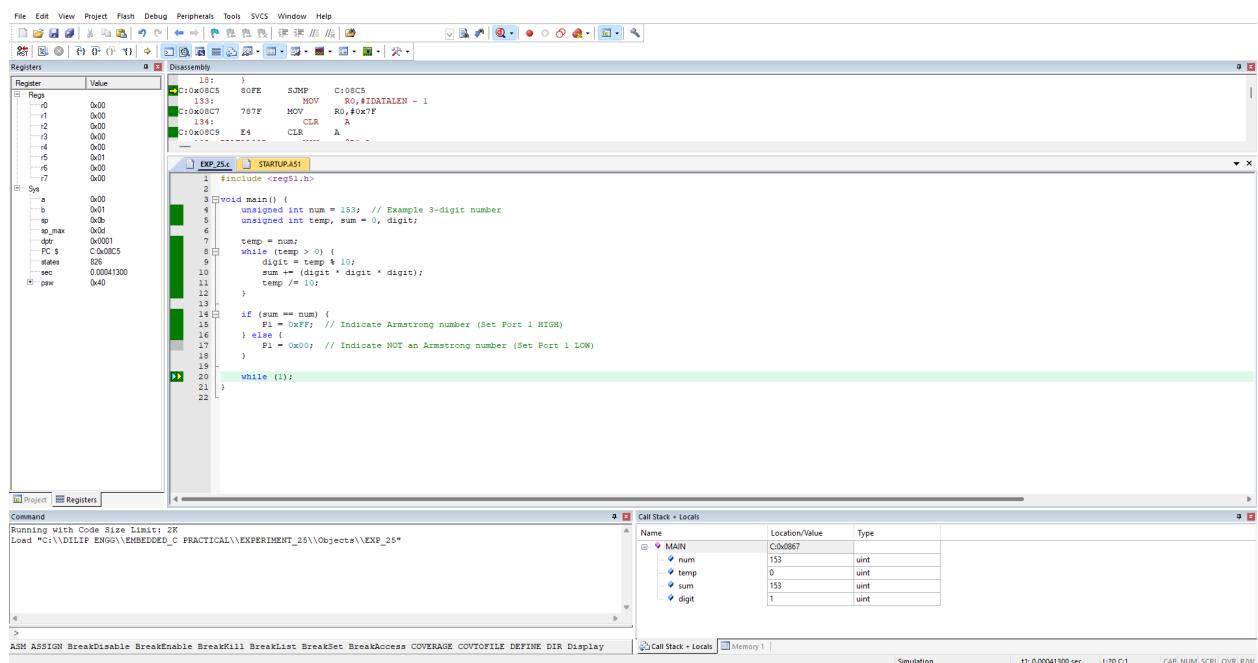
```
#include <reg51.h>

void main() {
    unsigned int num = 153; // Example 3-digit number
    unsigned int temp, sum = 0, digit;

    temp = num;
    while (temp > 0) {
        digit = temp % 10;
        sum += (digit * digit * digit);
        temp /= 10;
    }
}
```

```
if (sum == num) {  
    P1 = 0xFF; // Indicate Armstrong number (Set Port 1 HIGH)  
}  
else {  
    P1 = 0x00; // Indicate NOT an Armstrong number (Set Port 1 LOW)  
}  
  
while (1);  
}
```

## OUTPUT:



## RESULT:

To check whether a given 3-digit number is an Armstrong number is successful using a C program.

## **EXPERIMENT 26:**

### **Write a 8051 C program to swap the contents of two memory locations using a temporary variable**

#### **AIM:**

To swap the contents of two memory locations using a temporary variable by a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char *mem1 = (unsigned char *) 0x30; // Memory location 0x30
    unsigned char *mem2 = (unsigned char *) 0x31; // Memory location 0x31
    unsigned char temp;

    *mem1 = 0x55; // Example value at 0x30
    *mem2 = 0xAA; // Example value at 0x31

    temp = *mem1;
```

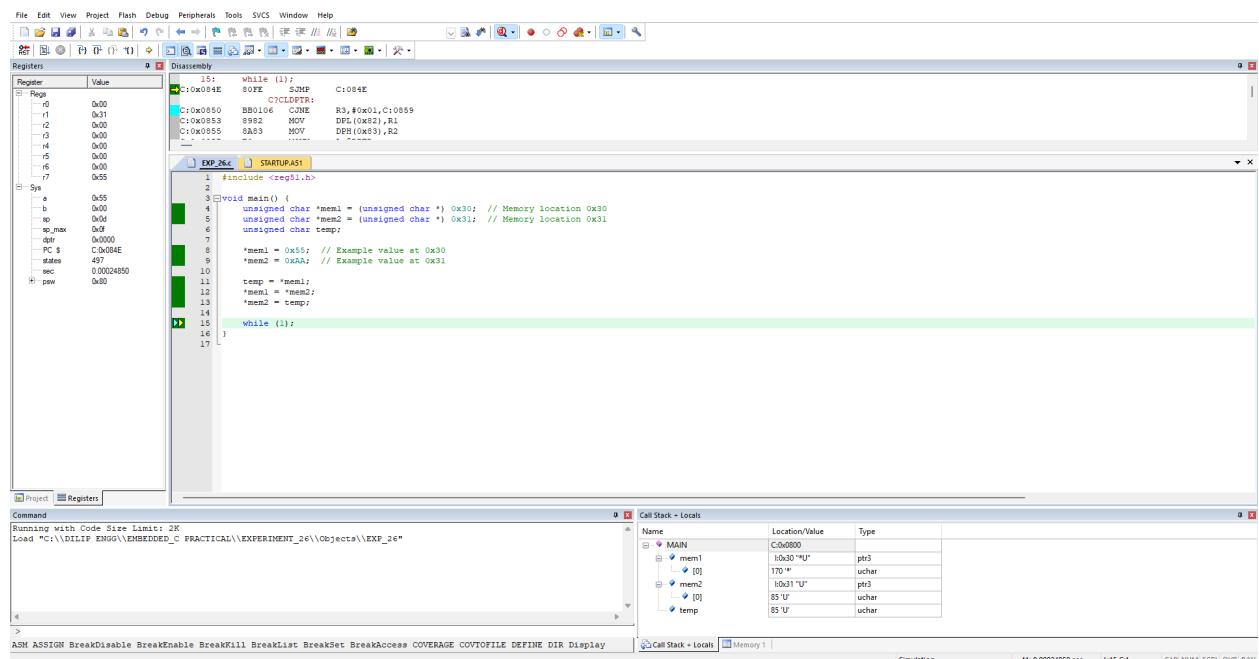
```

*mem1 = *mem2;
*mem2 = temp;

while (1);
}

```

## OUTPUT:



## RESULT:

To swap the contents of two memory locations using a temporary variable is successful by a C program.

## **EXPERIMENT 27:**

**Write a 8051 C program to shift the contents of an array to the left by one position (cyclic shift)**

### **AIM:**

To shift the contents of an array to the left by one position (cyclic shift) by a C program.

### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char arr[5] = {1, 2, 3, 4, 5};
    unsigned char temp, i;

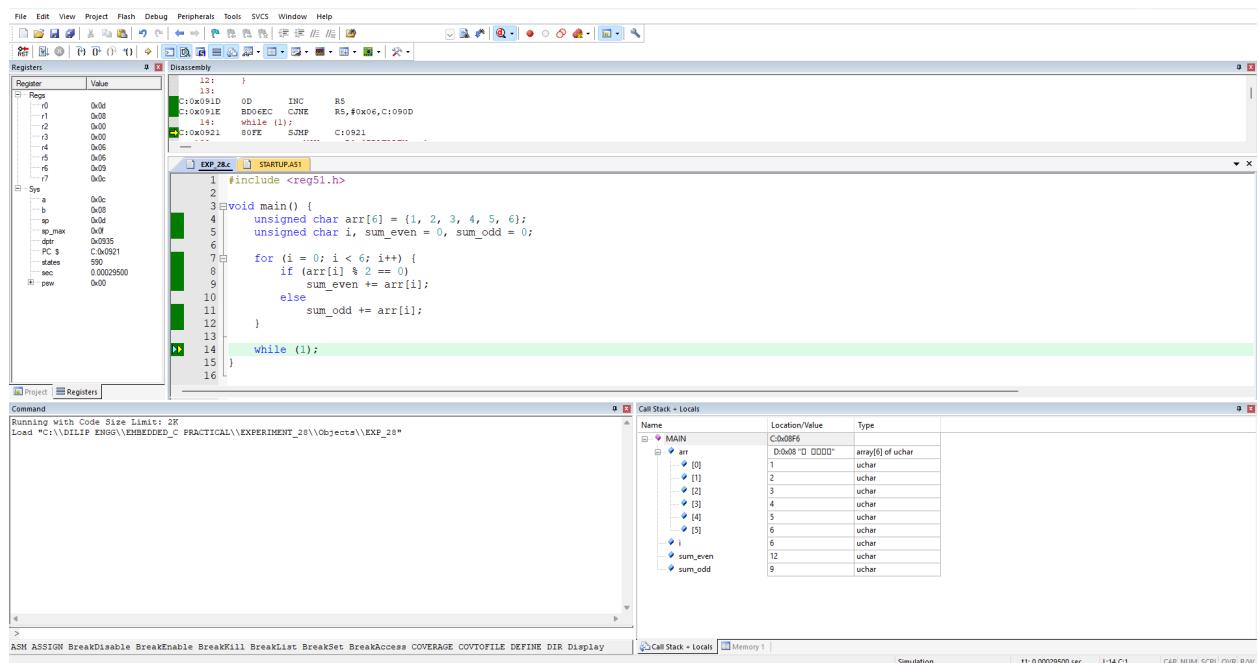
    temp = arr[0];
    for (i = 0; i < 4; i++) {
        arr[i] = arr[i + 1];
    }
    arr[4] = temp;
```

```

while (1);
}

```

## OUTPUT:



## RESULT:

To shift the contents of an array to the left by one position (cyclic shift) is successful by a C program.

## **EXPERIMENT 28:**

**Write a 8051 C program to find the sum of even and odd elements in an array separately**

### **AIM:**

To find the sum of even and odd elements in an array separately by a C program.

### **SOFTWARE REQUIRED:**

KEIL  $\mu$  Vision 5 Software

### **PROCEDURE:**

- Launch Keil  $\mu$ Vision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char arr[6] = {1, 2, 3, 4, 5, 6};
    unsigned char i, sum_even = 0, sum_odd = 0;

    for (i = 0; i < 6; i++) {
        if (arr[i] % 2 == 0)
            sum_even += arr[i];
        else
            sum_odd += arr[i];
    }
}
```

```

while (1);
}

```

## OUTPUT:

The screenshot shows a debugger interface with the following components:

- Registers:** Shows CPU registers (R0-R7, SP, PC, etc.) with their current values.
- Disassembly:** Shows assembly code corresponding to the C code. The assembly code includes instructions like INC, JNE, and CMP.
- C Code:** The C source code for the program, which defines an array arr[6] and calculates the sum of even and odd elements separately.
- Call Stack + Locals:** A table showing the call stack and local variable values. It lists variables like arr, i, sum\_even, and sum\_odd with their memory locations and values.
- Memory:** A table showing memory dump information.
- Command Line:** Shows the command line arguments and environment variables.

## RESULT:

To find the sum of even and odd elements in an array separately is successful by a C program.

## **EXPERIMENT 29:**

**Write a C program to convert a decimal number to binary (without built-in conversion functions)**

### **AIM:**

To convert a decimal number to binary (without build-in conversion function) by a C program.

### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned int num = 13; // Example decimal number
    unsigned char binary[16];
    unsigned char i = 0;

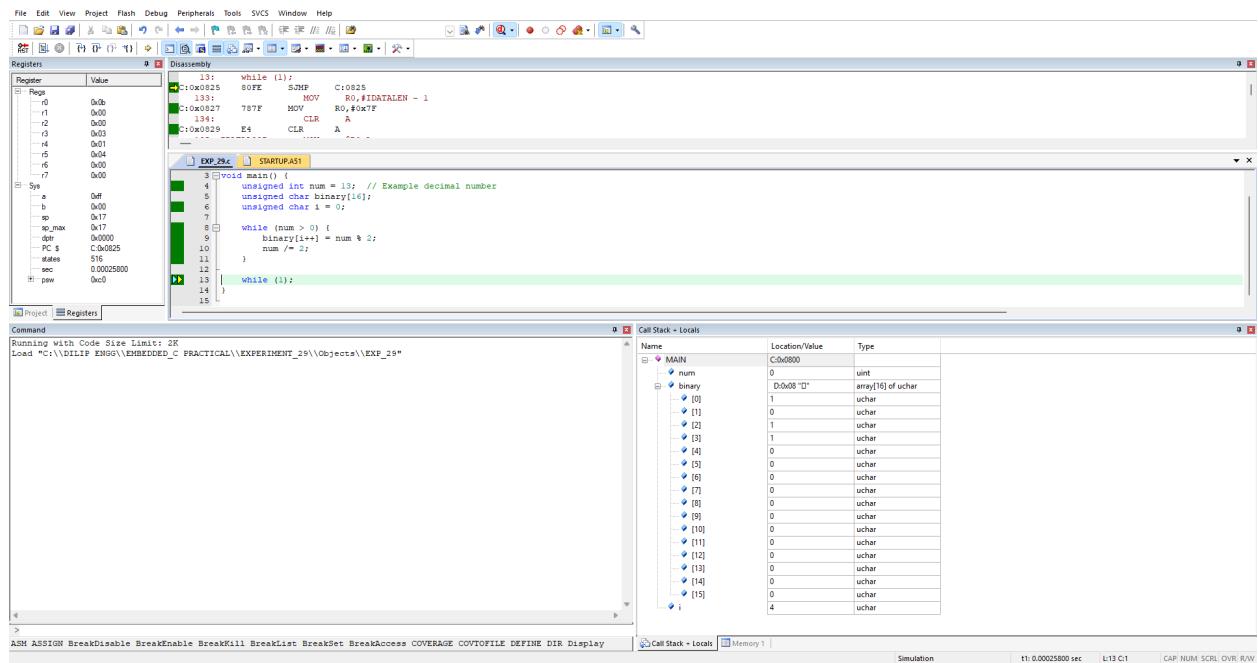
    while (num > 0) {
        binary[i++] = num % 2;
        num /= 2;
    }
}
```

```

while (1);
}

```

## OUTPUT:



## RESULT:

To convert a decimal number to binary (without a built-in conversion function) is successful by a C program.

## **EXPERIMENT 30:**

### **Write a 8051 C program to remove duplicate elements from an array of integers**

#### **AIM:**

To remove duplicate elements from an array of integers by a C program.

#### **SOFTWARE REQUIRED:**

KEIL µ Vision 5 Software

#### **PROCEDURE:**

- Launch Keil µVision 5 software.
- Navigate to the Project tab and select New Project, then save it in a desired folder.
- Choose AT89C51 as the target device and create a new C file.
- Write the C program and build the target.
- If errors occur, rectify them and rebuild the target.
- If no errors are detected, debug the program and verify the output.

#### **PROGRAM:**

```
#include <reg51.h>

void main() {
    unsigned char arr[6] = {1, 2, 2, 3, 4, 4};
    unsigned char result[6], i, j, k = 0, flag;

    for (i = 0; i < 6; i++) {
        flag = 0;
        for (j = 0; j < k; j++) {
            if (arr[i] == result[j]) {
                flag = 1;
                break;
            }
        }
        if (flag == 0) {
            result[k] = arr[i];
            k++;
        }
    }
}
```

```

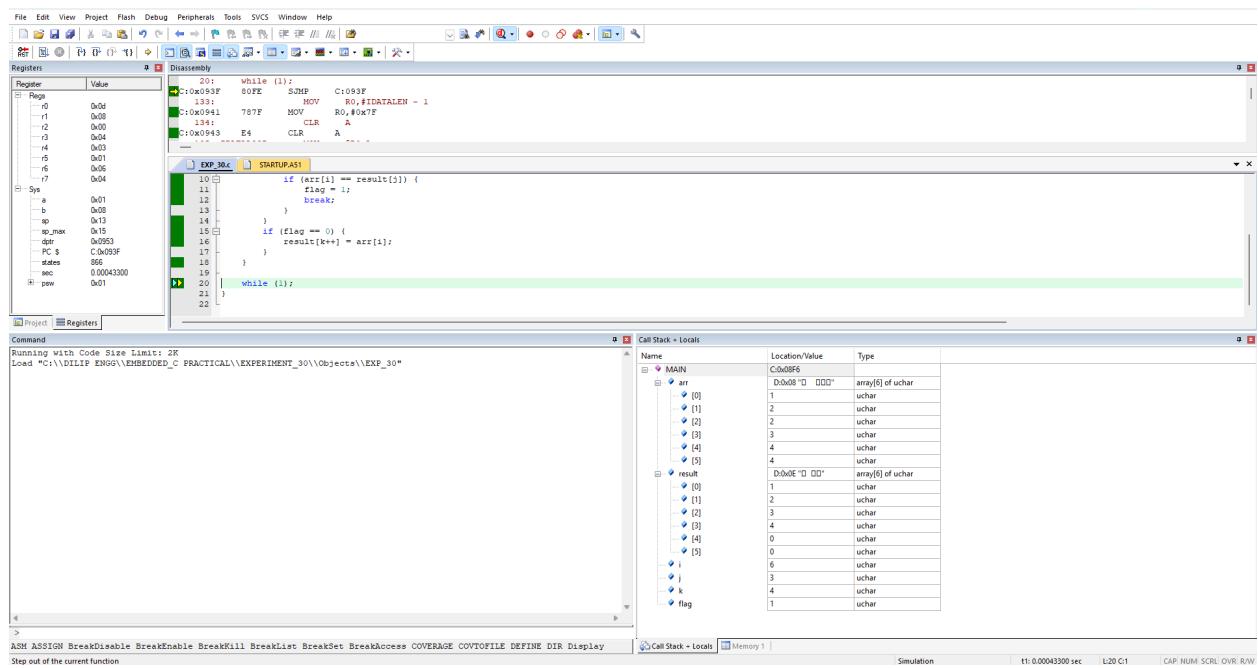
    }
}

if(flag == 0) {
    result[k++] = arr[i];
}
}

while(1);
}

```

## OUTPUT:



## RESULT:

To remove duplicate elements from an array of integers is successful by a C program.

