3. Write a program to identify particular number in the data using simulator

```
ORG 0x0000  ; Define the origin of the program

MOV R0, #25  ; Load the first number into register R0
MOV R1, #30  ; Load the second number into register R1

CJNE R0, R1, NotEqual  ; Compare R0 and R1, jump to NotEqual if they are not equal

; Code to execute if R0 is equal to R1
; (You can add your own code here)
; ...

SJMP End  ; Skip the NotEqual block and jump to the end

NotEqual:
; Code to execute if R0 is not equal to R1
; (You can add your own code here)
; ...

End:
NOP  ; No operation, can be used as a placeholder

END  ; End of the program
```

---

4. Write a program to identify smallest number in the data using EmbeddedC

```
#include <stdio.h>

int findSmallestNumber(int arr[], int size) {
    int smallest = arr[0];

    for (int i = 1; i < size; ++i) {
        if (arr[i] < smallest) {
            smallest = arr[i];
        }
    }

    return smallest;
}

int main() {
    int data[] = {23, 45, 12, 56, 78, 34, 9};
    int dataSize = sizeof(data) / sizeof(data[0]);

    int smallestNumber = findSmallestNumber(data, dataSize);

    printf("The smallest number in the data is: %d\n", smallestNumber);

    return 0;
}
```

---

5. Perform the communication between Raspberry Pl processor and Arduino using bluetooth

Raspberry PI (python code):

```
import serial
import time

# Replace '/dev/rfcomm0' with the Bluetooth serial port of your HC-05 module
ser = serial.Serial('/dev/rfcomm0', 9600, timeout=1)

try:
    while True:
        data = input("Enter data to send to Arduino: ")
        ser.write(data.encode())
        time.sleep(0.1)
        response = ser.readline().decode().strip()
        print("Arduino response:", response)

except KeyboardInterrupt:
    print("\nExiting program.")

finally:
    ser.close()
```

Arduino (Arduino Code):

```
void setup() {
  Serial.begin(9600);
}
```

```
void loop() {
  if (Serial.available() > 0) {
    char data = Serial.read();
    Serial.print("Received from Raspberry Pi: ");
    Serial.println(data);

    // Process the received data and send a response
    char response = processData(data);
    Serial.print("Sending response to Raspberry Pi: ");
    Serial.println(response);

    // Send the response back to the Raspberry Pi
    Serial.write(response);
  }
}

char processData(char input) {
  // Process the input data as needed
  // Here, we simply increment the ASCII value of the input character
  return input + 1;
}
```

---

6. Design the temperature level control using Raspberry Pl processor

Python Script:

```
import time
from w1thermsensor import W1ThermSensor
import RPi.GPIO as GPIO

# GPIO Pin for controlling the actuator (replace with your GPIO pin)
ACTUATOR_PIN = 17

# Set up GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(ACTUATOR_PIN, GPIO.OUT)

# Set up the temperature sensor
sensor = W1ThermSensor()

def read_temperature():
    return sensor.get_temperature()

def control_actuator(temperature_threshold, target_state):
    current_temperature = read_temperature()

    if current_temperature < temperature_threshold:
        GPIO.output(ACTUATOR_PIN, target_state)
    else:
        GPIO.output(ACTUATOR_PIN, not target_state)

if __name__ == "__main__":
```

```
try:
    # Set the temperature threshold and target state of the actuator (Heater, Cooler, etc.)
    TEMPERATURE_THRESHOLD = 25.0  # Replace with your desired threshold
    TARGET_STATE = GPIO.HIGH  # GPIO.HIGH for Heater, GPIO.LOW for Cooler

    while True:
        control_actuator(TEMPERATURE_THRESHOLD, TARGET_STATE)
        time.sleep(5)  # Adjust the interval as needed

except KeyboardInterrupt:
    print("\nExiting program.")

finally:
    GPIO.cleanup()
```

---

7. Write a program to ascending order in the data using Embedded C

```
#include <stdio.h>

void swap(int *x, int *y) {
   int temp = *x;
   *x = *y;
   *y = temp;
}

void bubbleSort(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            swap(&arr[j], &arr[j + 1]);
         }
      }
   }
}

int main() {
   int data[] = {23, 45, 12, 56, 78, 34, 9};
   int dataSize = sizeof(data) / sizeof(data[0]);

   printf("Original data: ");
   for (int i = 0; i < dataSize; i++) {
      printf("%d ", data[i]);
   }
```

```
    bubbleSort(data, dataSize);

    printf("\nData in ascending order: ");
    for (int i = 0; i < dataSize; i++) {
        printf("%d ", data[i]);
    }

    return 0;
}
```

---

10. Write a program to perform logical operations in the data using Embedded C

```
#include <stdio.h>

// Function to perform logical AND operation
unsigned int logicalAND(unsigned int x, unsigned int y) {
   return x & y;
}

// Function to perform logical OR operation
unsigned int logicalOR(unsigned int x, unsigned int y) {
   return x | y;
}

// Function to perform logical XOR operation
unsigned int logicalXOR(unsigned int x, unsigned int y) {
   return x ^ y;
}

// Function to perform logical NOT operation
unsigned int logicalNOT(unsigned int x) {
   return ~x;
}

int main() {
   unsigned int data1 = 0b11011010;  // Binary representation of a number
   unsigned int data2 = 0b10100101;  // Binary representation of another number

   printf("Data 1: %u (Binary: 0b%08b)\n", data1, data1);
```

```
    printf("Data 2: %u (Binary: 0b%08b)\n", data2, data2);

    printf("\nLogical AND: %u (Binary: 0b%08b)\n", logicalAND(data1, data2), logicalAND(data1, data2));
    printf("Logical OR : %u (Binary: 0b%08b)\n", logicalOR(data1, data2), logicalOR(data1, data2));
    printf("Logical XOR: %u (Binary: 0b%08b)\n", logicalXOR(data1, data2), logicalXOR(data1, data2));
    printf("Logical NOT: %u (Binary: 0b%08b)\n", logicalNOT(data1), logicalNOT(data1));

    return 0;
}
```

---

11. Write a program to perform arithmetic operations in 8051 using simulator

```
ORG 0x0000  ; Define the origin of the program

MOV A, #5    ; Load value 5 into accumulator A
MOV B, #3    ; Load value 3 into register B

; Addition
ADD A, B
; A now contains the result of A + B

; Subtraction
MOV B, #2    ; Load value 2 into register B for subtraction
SUBB A, B
; A now contains the result of (A + Carry) - B

; Multiplication
MOV A, #4    ; Load value 4 into accumulator A
MOV B, #6    ; Load value 6 into register B
MOV R2, A    ; Move A to register R2
MOV R3, B    ; Move B to register R3
MUL AB
; A and B now contain the 16-bit result of A * B

; Division
MOV A, #8    ; Load value 8 into accumulator A
MOV B, #2    ; Load value 2 into register B
DIV AB
; A now contains the quotient of A / B, and B contains the remainder
```

```
END  ; End of the program
```

----

12. Write a program to ALU operations in the data using Embedded C.

```
#include <stdio.h>

int main() {
   // Variables for arithmetic operations
   int a = 5;
   int b = 3;
   int result_add, result_sub, result_mul, result_div;

   // Addition
   result_add = a + b;

   // Subtraction
   result_sub = a - b;

   // Multiplication
   result_mul = a * b;

   // Division
   result_div = a / b;

   // Print results
   printf("Addition: %d\n", result_add);
   printf("Subtraction: %d\n", result_sub);
   printf("Multiplication: %d\n", result_mul);
   printf("Division: %d\n", result_div);
   return 0;
}
```

15. Write a program to ALU operations in the data using simulator.

```
ORG 0x0000  ; Define the origin of the program

MOV A, #5    ; Load value 5 into accumulator A
MOV B, #3    ; Load value 3 into register B

; Addition
ADD A, B
; A now contains the result of A + B

; Subtraction
MOV B, #2    ; Load value 2 into register B for subtraction
SUBB A, B
; A now contains the result of (A + Carry) - B

; Logical AND
MOV A, #0xAA ; Binary: 10101010
MOV B, #0x55 ; Binary: 01010101
ANL A, B
; A now contains the result of A AND B

; Logical OR
MOV A, #0xAA ; Binary: 10101010
MOV B, #0x55 ; Binary: 01010101
ORL A, B
; A now contains the result of A OR B

; Logical XOR
```

```
MOV A, #0xAA ; Binary: 10101010
MOV B, #0x55 ; Binary: 01010101
XRL A, B
; A now contains the result of A XOR B

; Logical NOT
MOV A, #0xAA ; Binary: 10101010
CPL A
; A now contains the result of NOT A

END  ; End of the program
```

---

17. Write a program to swap the data between memory and particular register.

```
ORG 0x0000  ; Define the origin of the program

MOV A, #55   ; Load value 55 into accumulator A
MOV R0, #0x30 ; Specify the memory address (replace with your desired address)
MOV @R0, A    ; Store the content of A at the memory address specified by R0

MOV A, @R0   ; Load the content from the memory address specified by R0 into A
MOV R1, A    ; Move the content of A to register R1

MOV A, #77   ; Load value 77 into accumulator A
MOV @R0, A   ; Store the content of A at the memory address specified by R0

MOV A, R1    ; Move the content of register R1 to A
MOV R2, A    ; Move the content of A to register R2

END  ; End of the program
```

---

18. Write a program to identify particular number in the data using simulator

```
ORG 0x0000  ; Define the origin of the program

MOV R0, #0   ; Initialize index to 0
MOV A, #42   ; Specify the number to search for
MOV DPTR, #Array ; Load the address of the array (replace Array with the actual address)

Search_Loop:
   MOV A, @DPTR ; Load the current element of the array into A
   INC DPTR     ; Move to the next element of the array
   INC R0       ; Increment the index

   ; Compare the current element with the specified number
   CJNE A, #42, Search_Loop ; If not equal, continue the search

   ; If equal, the index is stored in register R0
   ; You can use R0 as needed (e.g., store it in another register or memory location)

END  ; End of the program

Array: ; Replace this with the actual data array
DB 10, 20, 30, 40, 42, 50, 60  ; Example array
```