PRIYANKA G

BU22EECE0100446

HANDS ON ACTIVITY:

EMBEEDED SYSTEM FLOWCHART OF 7 PROGRAMS:

01. Write a program to count no. of bits which are set in given binary pattern2?

CODE

```
def count_set_bits(binary_pattern):
    # Convert the binary string to an integer
    number = int(binary_pattern, 2)

    # Initialize the count of set bits
    count = 0

    # Iterate through each bit of the integer
    while number:
        # Increment count if the least significant bit is set
        count += number & 1

        # Right shift the number to process the next bit
        number >>= 1

    return count

# Example usage
binary_pattern = "11010101"
set_bits_count = count_set_bits(binary_pattern)
```

print(f"The number of set bits in the binary pattern {binary_pattern} is {set_bits_count}.")The number of set bits in the binary pattern 11010101 is 5.

02.Write a program to set 5th and 12th bits in a 16-bit unsigned integer

CODE:

```
def set_bits(number, positions):
    # Iterate through the positions and set the corresponding bits
    for pos in positions:
        number |= (1 << pos)
    return number

# Example usage
number = 0b0000000000000000  # 16-bit unsigned integer with all bits set to 0
positions_to_set = [5, 12]

# Set the 5th and 12th bits
new_number = set_bits(number, positions_to_set)

# Print the results
print(f"Original number in binary: {bin(number)}")
print(f"Number after setting 5th and 12th bits in binary: {bin(new_number)}")
print(f"New number in decimal: {new_number}")
```

03.Write a program to clear 6th and 19th bits in a 32-bit unsigned integer

CODE:

```
def clear_bits(n, positions):
    # Create a mask with 1s everywhere except for the positions we want to clear
    mask = ~((1 << positions[0]) | (1 << positions[1]))
```

```
    # Apply the mask to clear the specified bits
    result = n & mask
    return result

# 32-bit unsigned integer example
n = 0b11111111111111111111111111111111  # Example input with all bits set to 1
positions = [6, 19]  # Positions of the bits to clear (0-indexed from the right)

# Clear the specified bits
result = clear_bits(n, positions)

# Print the result
print(f"Original: {n:032b}")
print(f"Modified: {result:032b}")

n = 0b11111111111111111111111111111111  # Example input with all bits set to 1
positions = [6, 19]  # Positions of the bits to clear (0-indexed from the right)

result = clear_bits(n, positions)

print(f"Original: {n:032b}")
print(f"Modified: {result:032b}")
```

OUTPUT:

Original: 11111111111111111111111111111111

Modified: 11111111111101111111011111111111

04. Write a program to flip even positioned bits in a 16-bit unsigned integer An IP Address will be in the form of "a. b, c. d" format, where a, b, c, d will be in the range of 0-255. Given a, b, c, d values (or string format) pack them into 32-bit unsigned integer

CODE:

```
def flip_even_bits(n):
    # Create a mask with 1s in even positions (0, 2, 4, ..., 14)
    mask = 0b0101010101010101  # 16-bit mask with 1s at even positions
    # XOR the number with the mask to flip the even bits
    result = n ^ mask
    return result

def pack_ip_address(a, b, c, d):
    # Shift and combine the IP address components into a 32-bit integer
    packed_ip = (a << 24) | (b << 16) | (c << 8) | d
    return packed_ip

# Task 1: Flip even positioned bits in a 16-bit unsigned integer
n = 0b1010101010101010  # Example input
flipped_result = flip_even_bits(n)
print(f"Original 16-bit number: {n:016b}")
print(f"Modified 16-bit number: {flipped_result:016b}")

# Task 2: Pack IP address into a 32-bit unsigned integer
a, b, c, d = 192, 168, 1, 1  # Example input
```

```
packed_ip = pack_ip_address(a, b, c, d)
print(f"Packed IP (Binary): {packed_ip:032b}")
print(f"Packed IP (Decimal): {packed_ip}")
```

05. Given an unsigned 32-bit integer holding packed IPv4 address, convert it into "a. b. c. d" format.

CODE:

```
def pack_ip_address(a, b, c, d):
    # Shift and combine the IP address components into a 32-bit integer
    packed_ip = (a << 24) | (b << 16) | (c << 8) | d
    return packed_ip

def unpack_ip_address(packed_ip):
    # Extract each 8-bit component from the 32-bit packed IP address
    a = (packed_ip >> 24) & 0xFF
    b = (packed_ip >> 16) & 0xFF
    c = (packed_ip >> 8) & 0xFF
    d = packed_ip & 0xFF
    return f"{a}.{b}.{c}.{d}"

# Example usage
a, b, c, d = 192, 168, 1, 1  # Example input
packed_ip = pack_ip_address(a, b, c, d)
print(f"Packed IP (Binary): {packed_ip:032b}")
print(f"Packed IP (Decimal): {packed_ip}")
```

```
unpacked_ip = unpack_ip_address(packed_ip)
print(f"Unpacked IP: {unpacked_ip}")
```

06. Convert MAC address into 48-bit binary pattern

CODE:

```
def pack_ip_address(a, b, c, d):
    # Shift and combine the IP address components into a 32-bit integer
    packed_ip = (a << 24) | (b << 16) | (c << 8) | d
    return packed_ip

def unpack_ip_address(packed_ip):
    # Extract each 8-bit component from the 32-bit packed IP address
    a = (packed_ip >> 24) & 0xFF
    b = (packed_ip >> 16) & 0xFF
    c = (packed_ip >> 8) & 0xFF
    d = packed_ip & 0xFF
    return f"{a}.{b}.{c}.{d}"

# Example usage
a, b, c, d = 192, 168, 1, 1  # Example input
packed_ip = pack_ip_address(a, b, c, d)
print(f"Packed IP (Binary): {packed_ip:032b}")
print(f"Packed IP (Decimal): {packed_ip}")

unpacked_ip = unpack_ip_address(packed_ip)
print(f"Unpacked IP: {unpacked_ip}")
```