Abhang Sudhir Pawar

24BDS054

def booths\_algorithm(multiplicand, multiplier, bit\_length=8): """ Implements Booth's multiplication algorithm for signed binary numbers in two's complement.

Args:  
 multiplicand: The multiplicand as a decimal integer  
 multiplier: The multiplier as a decimal integer  
 bit\_length: Number of bits to use for representation (default: 8)  
   
Returns:  
 The product of multiplicand and multiplier  
"""  
# Ensure the numbers fit within the bit\_length (handle overflows)  
max\_value = 2\*\*(bit\_length-1) - 1  
min\_value = -2\*\*(bit\_length-1)  
  
if multiplicand > max\_value or multiplicand < min\_value or \  
 multiplier > max\_value or multiplier < min\_value:  
 print(f"Warning: Numbers may not fit in {bit\_length}-bit representation")  
 print(f"Valid range for {bit\_length}-bit: {min\_value} to {max\_value}")  
  
# Step 1: Initialize values  
A = 0 # Accumulator  
  
# Handle two's complement representation properly  
M = multiplicand & ((1 << bit\_length) - 1) # Multiplicand in bit\_length bits  
Q = multiplier & ((1 << bit\_length) - 1) # Multiplier in bit\_length bits  
Q\_minus\_1 = 0 # Extra bit initialized to 0  
  
# Helper function to convert from 2's complement to signed decimal  
def to\_signed\_decimal(value, bits):  
 if (value >> (bits - 1)) & 1: # If sign bit is set  
 return value - (1 << bits)  
 return value  
  
print(f"Initial values:")  
print(f"A = {bin(A)[2:].zfill(bit\_length)} (decimal: {to\_signed\_decimal(A, bit\_length)})")  
print(f"M = {bin(M)[2:].zfill(bit\_length)} (decimal: {multiplicand})")  
print(f"Q = {bin(Q)[2:].zfill(bit\_length)} (decimal: {multiplier})")  
print(f"Q\_-1 = {Q\_minus\_1}")  
print("-" \* 50)  
  
# Step 2: Iterate for bit\_length times  
for i in range(bit\_length):  
 print(f"Iteration {i+1}:")  
   
 # Examine the rightmost bit of Q and Q\_minus\_1  
 Q\_0 = Q & 1 # Rightmost bit of Q  
   
 # Based on the two bits, perform operation  
 if Q\_0 == 1 and Q\_minus\_1 == 0:  
 # If 10, subtract M from A  
 print(f"Q\_0 Q\_-1 = 10, subtracting M from A")  
 A = (A - (multiplicand & ((1 << bit\_length) - 1))) & ((1 << bit\_length) - 1)  
 elif Q\_0 == 0 and Q\_minus\_1 == 1:  
 # If 01, add M to A  
 print(f"Q\_0 Q\_-1 = 01, adding M to A")  
 A = (A + (multiplicand & ((1 << bit\_length) - 1))) & ((1 << bit\_length) - 1)  
 else:  
 print(f"Q\_0 Q\_-1 = {Q\_0}{Q\_minus\_1}, no arithmetic operation")  
   
 print(f"After operation: A = {bin(A)[2:].zfill(bit\_length)} (decimal: {to\_signed\_decimal(A, bit\_length)})")  
   
 # Arithmetic shift right (preserves sign bit)  
 # Save the rightmost bit of Q before shifting  
 Q\_minus\_1 = Q & 1  
   
 # Check if A has sign bit set (for arithmetic right shift)  
 sign\_bit = (A >> (bit\_length - 1)) & 1  
   
 # Perform the right shift for Q, incorporating A's LSB  
 Q = (Q >> 1) | ((A & 1) << (bit\_length - 1))  
   
 # Perform arithmetic right shift for A (preserving sign bit)  
 A = (A >> 1) | (sign\_bit << (bit\_length - 1))  
   
 # Get the current partial result  
 partial\_result = (A << bit\_length) | Q  
 signed\_partial = partial\_result  
 if (signed\_partial >> (2 \* bit\_length - 1)) & 1:  
 signed\_partial = signed\_partial - (1 << (2 \* bit\_length))  
   
 print(f"After shift: A = {bin(A)[2:].zfill(bit\_length)} (decimal: {to\_signed\_decimal(A, bit\_length)}), "  
 f"Q = {bin(Q)[2:].zfill(bit\_length)} (decimal: {to\_signed\_decimal(Q, bit\_length)}), "  
 f"Q\_-1 = {Q\_minus\_1}")  
 print(f"Current partial result: {signed\_partial}")  
 print("-" \* 50)  
  
# Step 3: The result is in A and Q concatenated (2\*bit\_length bits)  
result\_binary = (A << bit\_length) | Q  
  
# Check if the result is negative (MSB of the 2\*bit\_length result is 1)  
result\_decimal = result\_binary  
if (result\_binary >> (2 \* bit\_length - 1)) & 1:  
 # Convert from 2's complement to negative decimal  
 result\_decimal = result\_binary - (1 << (2 \* bit\_length))  
  
# Return both binary and decimal versions  
return result\_binary, result\_decimal

def main(): """ Main function to get user input and run Booth's algorithm """ print("Booth's Multiplication Algorithm") print("--------------------------------")

try:  
 # Get input from user  
 multiplicand = int(input("Enter multiplicand: "))  
 multiplier = int(input("Enter multiplier: "))  
   
 # Use fixed bit length of 8  
 bit\_length = 8  
   
 # Calculate the result  
 print("\nExecuting Booth's Algorithm...\n")  
 result\_binary, result\_decimal = booths\_algorithm(multiplicand, multiplier, bit\_length)  
   
 # Show final result  
 print("\nFinal Result:")  
 print(f"Binary result: {bin(result\_binary)[2:].zfill(2\*bit\_length)}")  
 print(f"Decimal result: {multiplicand} × {multiplier} = {result\_decimal}")  
 expected = multiplicand \* multiplier  
 print(f"Python's built-in multiplication: {expected}")  
   
 if result\_decimal == expected:  
 print("✓ Results match!")  
 else:  
 print("⚠ Results don't match. This could be due to bit length limitations.")  
   
except ValueError:  
 print("Error: Please enter valid integer numbers.")  
except KeyboardInterrupt:  
 print("\nOperation cancelled by user.")  
except Exception as e:  
 print(f"An error occurred: {e}")

if *name* == "*main*": main()