1.Implement a program (MIPS Assembly) which multiplies user input by 10 using only bit shift operations and addition. Check to see if your program is correct by using the mult and mflo operators. Your program should include a proper and useful prompt for input, and print the results in a meaningful manner.

.data

prompt: .asciiz "Enter an integer: "

resultMsg: .asciiz "The result of multiplying by 10 using shift operations is: "

checkMsg: .asciiz "The result of multiplying by 10 using mult operator is: "

.text

main:

# Print the prompt

li $v0, 4 # Load the print string syscall

la $a0, prompt # Load address of the prompt

syscall # Print the prompt

# Read an integer from the user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t0, $v0 # Move the input to $t0

# Perform the multiplication by 10 using shift operations

sll $t1, $t0, 3 # $t1 = $t0 << 3 (multiply by 8)

sll $t2, $t0, 1 # $t2 = $t0 << 1 (multiply by 2)

add $t3, $t1, $t2 # $t3 = $t1 + $t2 (multiply by 10)

# Print the result message for shift operations

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print the result of shift operations

move $a0, $t3 # Move the result to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Perform the multiplication by 10 using mult and mflo

li $t4, 10 # Load 10 into $t4

mult $t0, $t4 # Multiply $t0 by $t4

mflo $t5 # Move the lower 32 bits of the result to $t5

# Print the result message for mult operator

li $v0, 4 # Load the print string syscall

la $a0, checkMsg # Load address of the check message

syscall # Print the check message

# Print the result of mult operator

move $a0, $t5 # Move the result to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Exit the program

li $v0, 10 # Load the exit syscall

syscall # Exit the program

2. Write programs (MIPS Assembly) to evaluate the following expressions. The user should enter the variables, and the program should print back an answer. Prompt the user for all variables in the expression, and print the results in a meaningful manner.

a) 5x + 3y + z

.data

prompt\_x: .asciiz "Enter value for x: "

prompt\_y: .asciiz "Enter value for y: "

prompt\_z: .asciiz "Enter value for z: "

resultMsg: .asciiz "The result of 5x + 3y + z is: "

.text

main:

# Prompt for x

li $v0, 4 # Load the print string syscall

la $a0, prompt\_x # Load address of the prompt for x

syscall # Print the prompt

# Read integer x from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t0, $v0 # Move the input to $t0 (x)

# Prompt for y

li $v0, 4 # Load the print string syscall

la $a0, prompt\_y # Load address of the prompt for y

syscall # Print the prompt

# Read integer y from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t1, $v0 # Move the input to $t1 (y)

# Prompt for z

li $v0, 4 # Load the print string syscall

la $a0, prompt\_z # Load address of the prompt for z

syscall # Print the prompt

# Read integer z from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t2, $v0 # Move the input to $t2 (z)

# Compute 5x using shift and add

sll $t3, $t0, 2 # $t3 = x << 2 (4x)

add $t3, $t3, $t0 # $t3 = 4x + x (5x)

# Compute 3y using shift and add

sll $t4, $t1, 1 # $t4 = y << 1 (2y)

add $t4, $t4, $t1 # $t4 = 2y + y (3y)

# Compute 5x + 3y

add $t5, $t3, $t4 # $t5 = 5x + 3y

# Compute 5x + 3y + z

add $t6, $t5, $t2 # $t6 = 5x + 3y + z

# Print the result message

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print the result

move $a0, $t6 # Move the result to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Exit the program

li $v0, 10 # Load the exit syscall

syscall # Exit the program

b) (

.data

prompt\_x: .asciiz "Enter value for x: "

prompt\_y: .asciiz "Enter value for y: "

prompt\_z: .asciiz "Enter value for z: "

resultMsg: .asciiz "The result of ((5x + 3y + z) / 2) \* 3 is: "

.text

main:

# Prompt for x

li $v0, 4 # Load the print string syscall

la $a0, prompt\_x # Load address of the prompt for x

syscall # Print the prompt

# Read integer x from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t0, $v0 # Move the input to $t0 (x)

# Prompt for y

li $v0, 4 # Load the print string syscall

la $a0, prompt\_y # Load address of the prompt for y

syscall # Print the prompt

# Read integer y from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t1, $v0 # Move the input to $t1 (y)

# Prompt for z

li $v0, 4 # Load the print string syscall

la $a0, prompt\_z # Load address of the prompt for z

syscall # Print the prompt

# Read integer z from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t2, $v0 # Move the input to $t2 (z)

# Compute 5x using shift and add

sll $t3, $t0, 2 # $t3 = x << 2 (4x)

add $t3, $t3, $t0 # $t3 = 4x + x (5x)

# Compute 3y using shift and add

sll $t4, $t1, 1 # $t4 = y << 1 (2y)

add $t4, $t4, $t1 # $t4 = 2y + y (3y)

# Compute 5x + 3y

add $t5, $t3, $t4 # $t5 = 5x + 3y

# Compute 5x + 3y + z

add $t6, $t5, $t2 # $t6 = 5x + 3y + z

# Compute (5x + 3y + z) / 2

sra $t7, $t6, 1 # $t7 = $t6 >> 1 ((5x + 3y + z) / 2)

# Compute ((5x + 3y + z) / 2) \* 3

sll $t8, $t7, 1 # $t8 = $t7 << 1 (2 \* (5x + 3y + z) / 2)

add $t8, $t8, $t7 # $t8 = $t8 + $t7 (3 \* (5x + 3y + z) / 2)

# Print the result message

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print the result

move $a0, $t8 # Move the result to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Exit the program

li $v0, 10 # Load the exit syscall

syscall # Exit the program

c) x3 + 2x2 + 3x + 4

.data

prompt\_x: .asciiz "Enter value for x: "

resultMsg: .asciiz "The result of x^3 + 2x^2 + 3x + 4 is: "

.text

main:

# Prompt for x

li $v0, 4 # Load the print string syscall

la $a0, prompt\_x # Load address of the prompt for x

syscall # Print the prompt

# Read integer x from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t0, $v0 # Move the input to $t0 (x)

# Compute x^3 (x \* x \* x)

mul $t1, $t0, $t0 # $t1 = x \* x (x^2)

mul $t2, $t1, $t0 # $t2 = $t1 \* x (x^3)

# Compute 2x^2 (2 \* (x \* x))

mul $t3, $t1, 2 # $t3 = $t1 \* 2 (2x^2)

# Compute 3x (3 \* x)

mul $t4, $t0, 3 # $t4 = x \* 3 (3x)

# Add x^3 + 2x^2 + 3x + 4

add $t5, $t2, $t3 # $t5 = x^3 + 2x^2

add $t6, $t5, $t4 # $t6 = $t5 + 3x

addi $t7, $t6, 4 # $t7 = $t6 + 4

# Print the result message

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print the result

move $a0, $t7 # Move the result to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Exit the program

li $v0, 10 # Load the exit syscall

syscall # Exit the program

d) () \*y

.data

prompt\_x: .asciiz "Enter value for x: "

prompt\_y: .asciiz "Enter value for y: "

resultMsg: .asciiz "The result of (4x / 3) \* y is: "

.text

main:

# Prompt for x

li $v0, 4 # Load the print string syscall

la $a0, prompt\_x # Load address of the prompt for x

syscall # Print the prompt

# Read integer x from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t0, $v0 # Move the input to $t0 (x)

# Prompt for y

li $v0, 4 # Load the print string syscall

la $a0, prompt\_y # Load address of the prompt for y

syscall # Print the prompt

# Read integer y from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t1, $v0 # Move the input to $t1 (y)

# Compute 4x using shift

sll $t2, $t0, 2 # $t2 = x << 2 (4x)

# Compute 4x / 3

li $t3, 3 # Load the divisor 3 into $t3

div $t2, $t3 # Divide $t2 by $t3

mflo $t4 # Move the quotient to $t4 (4x / 3)

# Compute (4x / 3) \* y

mul $t5, $t4, $t1 # $t5 = $t4 \* $t1 ((4x / 3) \* y)

# Print the result message

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print the result

move $a0, $t5 # Move the result to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Exit the program

li $v0, 10 # Load the exit syscall

syscall # Exit the program

3. Write a program (MIPS Assembly) to retrieve two numbers from a user and swap those numbers using only the XOR operation. You should not use a temporary variable to store the numbers while swapping them. Your program should include a proper and useful prompt for input, and print the results in a meaningful manner.

.data

prompt\_a: .asciiz "Enter the first number (a): "

prompt\_b: .asciiz "Enter the second number (b): "

resultMsg: .asciiz "After swapping, the numbers are:\n"

result\_a: .asciiz "a: "

result\_b: .asciiz "b: "

.text

main:

# Prompt for a

li $v0, 4 # Load the print string syscall

la $a0, prompt\_a # Load address of the prompt for a

syscall # Print the prompt

# Read integer a from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t0, $v0 # Move the input to $t0 (a)

# Prompt for b

li $v0, 4 # Load the print string syscall

la $a0, prompt\_b # Load address of the prompt for b

syscall # Print the prompt

# Read integer b from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t1, $v0 # Move the input to $t1 (b)

# Swap a and b using XOR

xor $t0, $t0, $t1 # a = a ^ b

xor $t1, $t0, $t1 # b = a ^ b (original a)

xor $t0, $t0, $t1 # a = a ^ b (original b)

# Print the result message

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print the swapped value of a

li $v0, 4 # Load the print string syscall

la $a0, result\_a # Load address of the result a message

syscall # Print the result a message

move $a0, $t0 # Move the swapped value of a to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Print a newline for formatting

li $v0, 11 # Load the print character syscall

li $a0, 10 # ASCII code for newline

syscall # Print the newline

# Print the swapped value of b

li $v0, 4 # Load the print string syscall

la $a0, result\_b # Load address of the result b message

syscall # Print the result b message

move $a0, $t1 # Move the swapped value of b to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Print a newline for formatting

li $v0, 11 # Load the print character syscall

li $a0, 10 # ASCII code for newline

syscall # Print the newline

# Exit the program

li $v0, 10 # Load the exit syscall

syscall # Exit the program

4. Using only sll and srl, implement a program to check if a user input value is even or odd. The result should print out 0 if the number is even or 1 if the number is odd. Your program (MIPS Assembly) should include a proper and useful prompt for input, and print the results in a meaningful manner.

.data

prompt: .asciiz "Enter an integer: "

evenMsg: .asciiz "The number is even.\n"

oddMsg: .asciiz "The number is odd.\n"

resultMsg: .asciiz "Result (0 for even, 1 for odd): "

.text

main:

# Prompt for input

li $v0, 4 # Load the print string syscall

la $a0, prompt # Load address of the prompt

syscall # Print the prompt

# Read integer from user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t0, $v0 # Move the input to $t0

# Isolate LSB by shifting right and left

sll $t1, $t0, 31 # Shift left by 31 bits to isolate the LSB

srl $t1, $t1, 31 # Shift right by 31 bits to bring LSB to the LSB position

# Print the result message

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print the result (0 for even, 1 for odd)

move $a0, $t1 # Move the result to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

# Check if the number is even or odd and print the appropriate message

beqz $t1, print\_even # If $t1 is 0, branch to print\_even

b print\_odd # Otherwise, branch to print\_odd

print\_even:

li $v0, 4 # Load the print string syscall

la $a0, evenMsg # Load address of the even message

syscall # Print the even message

b exit # Exit the program

print\_odd:

li $v0, 4 # Load the print string syscall

la $a0, oddMsg # Load address of the odd message

syscall # Print the odd message

exit:

li $v0, 10 # Load the exit syscall

syscall # Exit the program

5. Implement a program (MIPS Assembly) to prompt the user for two numbers, the first being any number and the second a prime number. Return to the user a 0 if the second number is a prime factor for the first one, otherwise any number if it is not. For example, if the user enters 60 and 5 , the program returns 0 . If the user enters 62 and 5 , the program returns 2.

.data

prompt\_num1: .asciiz "Enter the first number: "

prompt\_prime: .asciiz "Enter the prime number: "

resultMsg: .asciiz "The result is: "

.text

main:

# Prompt for the first number

li $v0, 4 # Load the print string syscall

la $a0, prompt\_num1 # Load address of the prompt for the first number

syscall # Print the prompt

# Read the first number from the user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t0, $v0 # Move the input to $t0 (first number)

# Prompt for the prime number

li $v0, 4 # Load the print string syscall

la $a0, prompt\_prime # Load address of the prompt for the prime number

syscall # Print the prompt

# Read the prime number from the user

li $v0, 5 # Load the read integer syscall

syscall # Read the integer

move $t1, $v0 # Move the input to $t1 (prime number)

# Perform the division to find the remainder

div $t0, $t1 # Divide $t0 by $t1

mfhi $t2 # Move the remainder to $t2

# Check if the remainder is zero

beqz $t2, is\_factor # If remainder is zero, branch to is\_factor

b not\_factor # Otherwise, branch to not\_factor

is\_factor:

# Print the result message

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print 0 (second number is a prime factor)

li $a0, 0 # Load 0 into $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

b exit # Exit the program

not\_factor:

# Print the result message

li $v0, 4 # Load the print string syscall

la $a0, resultMsg # Load address of the result message

syscall # Print the result message

# Print the remainder (second number is not a prime factor)

move $a0, $t2 # Move the remainder to $a0

li $v0, 1 # Load the print integer syscall

syscall # Print the integer

exit:

# Exit the program

li $v0, 10 # Load the exit syscall

syscall # Exit the program