Київський національний університет

імені Т.Шевченка

**Звіт**

до лабораторної роботи №3

на тему:

«*Імітаційна модель математичного співпроцесора****»***

**Мета**

Метою даної лабораторної роботи є розробка програмної моделі співпроцесора, реалізація його імітаційної (тобто комп’ютерної) моделі та розробка представлення введеного у діалозі десяткового числа у експоненціальній формі у форматі ЧПТ.

**Основні принципи виконання роботи**

Нижче подані основні принципи написання, не вдаючись до детального розбору коду.

Варіант 8.10.11

Основні характеристики:

* Cтек з 6 регістрів
* Формат числа з плаваючою точкою (ЧПТ) має стандартну структуру: sEM де: s –біт знаку мантиси, E – біти характеристики, M – біти мантиси у прямому коді. Мантиса має ще один неявний біт для старшої одиниці нормалізованого представлення, яка є єдиним розрядом цілої частини; Наприклад, мій варіант 8x.10m.11V: формат ЧПТ 8 бітів для характеристики та 10 явних біти у мантисі плюс знаковий біт; 11 варіант із таблиці формул для обрахунку співпроцесором;
* Дані в регістрах подаються в форматі IEEE 754;
* (sin(ln(y) + 2) \* cos(x) – формула для обрахунку співпроцесором;
* 2 такти: 1-й) занесення поточної команди у регістр команди; 2-й) виконання команди;
* Регістр команд, регістр тактів та регістр стану(фіксується знак результату виконання команди);

Основні задачі, які було виконано:

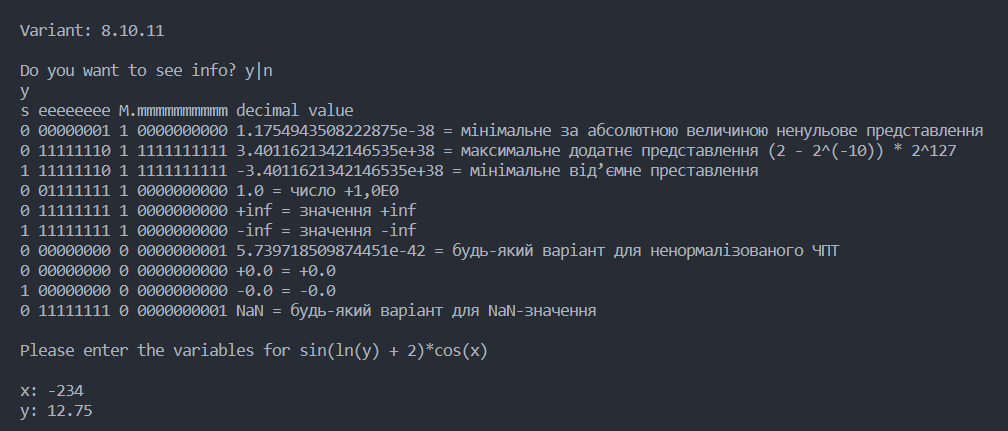
* Відрахунок тактів. Реалізувала натискання клавіші Enter на клавіатурі для переходу до наступного такту.
* Представлення числа в форматі IEEE 754
* Cтек;
* 8 команд, необхідних для вирішення формули, заданої варіантом;
* Опрацьовуються і такі значення: мінімальне за абсолютною величиною ненульове представлення, максимальне додатнє представлення, мінімальне від’ємне преставлення, число +1,0Е0, значення +∞, значення -∞, будь-який варіант для ненормалізованого ЧПТ, будь-який варіант для NaN-значення;

Послідовність виконання команд, для вирішення формули (sin(ln(y) + 2) \* cos(x)

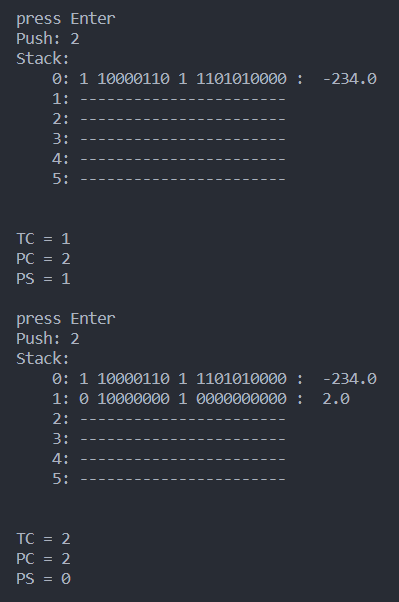
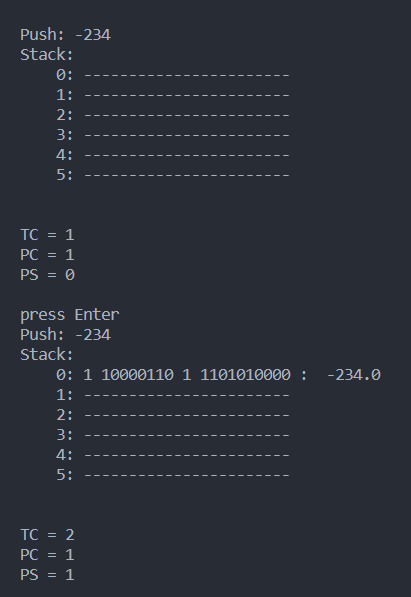
|  |  |  |  |
| --- | --- | --- | --- |
| Команди | Операнди | Формула | Стек |
| Добавити в стек | push(x) | x | x |
| Добавити в стек | push(2) | 2 | 2, x |
| Добавити в стек | push(y) | y | y, 2, x |
| Логарифм | ln() | ln(y) | ln(y), 2, x |
| Додавання | plus() | ln(y) + 2 | ln(y) + 2, x |
| Синус | sinn() | sin(ln(y) + 2) | sin(ln(y) + 2), x |
| Поміняти останні два  елементи стеку місцями | reverse() | x | x, sin(ln(y) + 2) |
| Косинус | coss() | cos(x) | cos(x), sin(ln(y) + 2) |
| Mult | mult() | (sin(ln(y) + 2) \* cos(x) | (sin(ln(y) + 2) \* cos(x) |

Команда ln() використовує команду pop() в середині себе, щоб забрати останній надходжений у стек елемений. А після виконання використовує команду push(), щоб добавити результат у стек. Команди coss() та sinn() працюють аналогічно. А команди plus() та mult() використовують команду pop() двічі, щоб отримати 1 та 2 операнд для роботи. Після виконання операції, вони аналогічно визивають команду push(). А команда reverse визиває 2 pop() та 2 push().

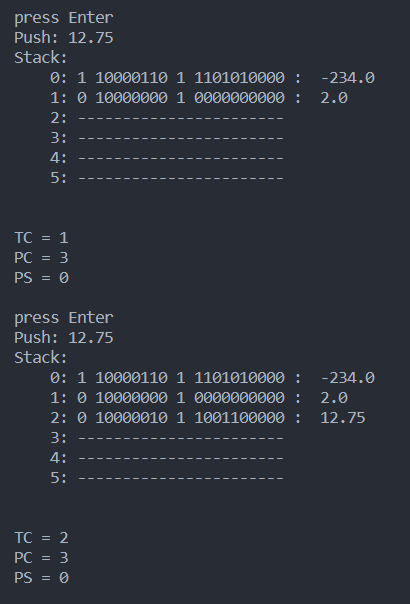
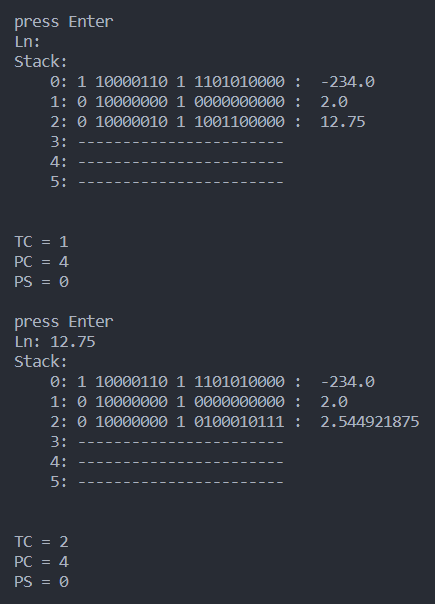
**Лістинг програми**

****

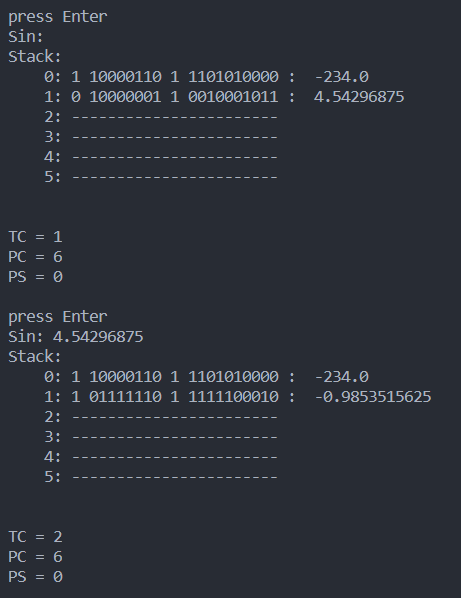
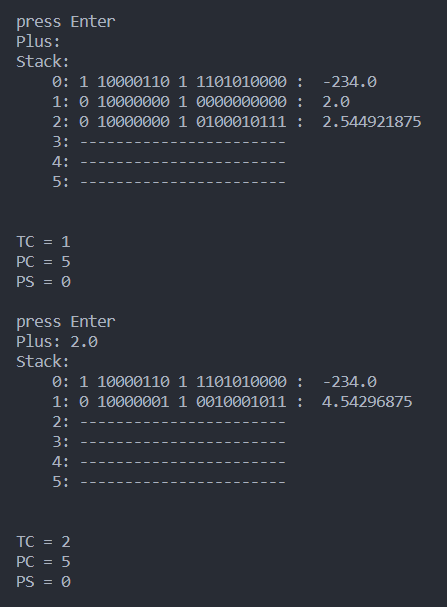
1. **2)**

****

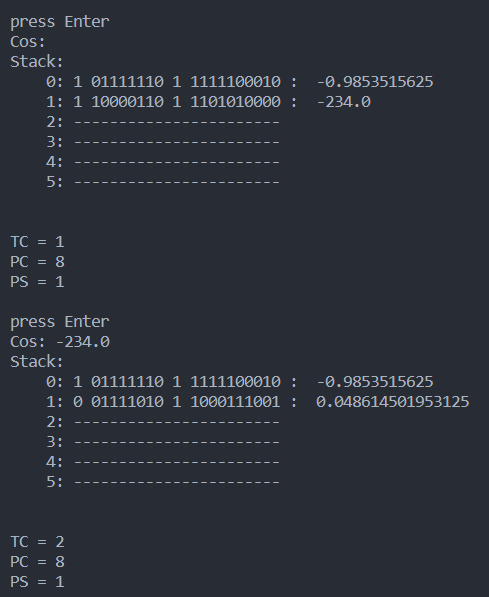
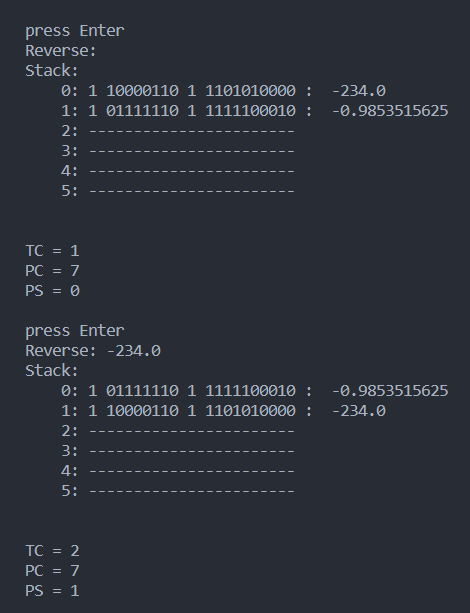
**3) 4)**

**** ****

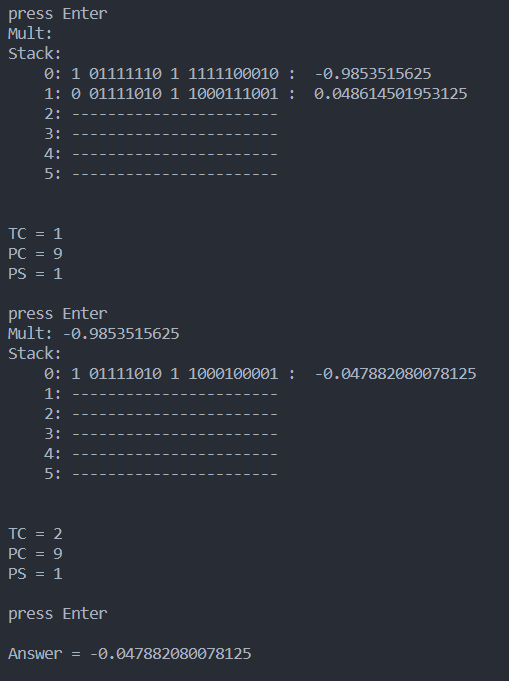
**5) 6)**

****

**7) 8)**

****

**9)**

****

**Висновок**

Математичний співпроцесор, або модуль (блок) операцій з рухомою комою (англ. Floating point unit, FPU) — співпроцесор для розширення системи команд центрального процесора комп'ютера командами для здійснення операцій над числами з рухомою комою.

Доволі легко на апаратному рівні порівнювати два дійсних числа, використовуючи ЧПТ (спочатку проглядається знак, потім характеристика та мантиса). Також це дозволяє проводити математичні операції з дійсними числами.

А отже, ЧПТ є оптимальним для представлення дійсних чисел.**Додаток. Код програми**

*# 8.10.11*

*# sin(ln(y) + 2) \* cos(x)*

*from* math *import* sin, cos, log

stack = []

PC = 0  *# command number*

TC = 1  *# tact*

PS = 0  *# sign*

STACK\_SIZE = 6

EXPONENT\_WIDTH = 8

MANTISS\_WIDTH = 10

WIDTH = 1 + EXPONENT\_WIDTH + MANTISS\_WIDTH

EXP\_SHIFT = 2 \*\* (EXPONENT\_WIDTH - 1) - 1

MAXIMUM\_EXP = 2 \*\* EXPONENT\_WIDTH - 1 - EXP\_SHIFT

MINIMUM\_EXP = -EXP\_SHIFT

*# decor*

*# =====================*

def seeStack():

    print('Stack:')

*for* i *in* range(STACK\_SIZE):

*if* i < len(stack):

            string = seeBin(stack[i]) + " :  " + str(seeNumber(stack[i]))

            print(f'    {i}: {string}')

*else*:

            print(f"    {i}: -----------------------")

    print("\n")

def wait():

    input("press Enter")

def draw():

    seeStack()

    print(f"TC = {TC % 2 + 1}")

    print(f"PC = {PC}")

    print(f"PS = {PS}\n")

def firstTact(*operation*, *str*=" "):

    global TC, PC

    PC += 1

    TC += 1

    print(f"{*operation*}: {*str*}")

    draw()

    wait()

def secondTact(*operation*, *num*):

    global TC, PS

    TC += 1

*if* float(*num*) < 0:

        PS = 1

*else*:

        PS = 0

    print(f"{*operation*}: {*num*}")

    draw()

    wait()

def seeInfo():

    print(f's {"e" \* EXPONENT\_WIDTH} M.{"m" \* MANTISS\_WIDTH} decimal value')

    num = [0] + [0] \* (EXPONENT\_WIDTH - 1) + [1] + [0] \* MANTISS\_WIDTH

    print(f'{seeBin(num)} {seeNumber(num)} = мінімальне за абсолютною величиною ненульове представлення')

    num = [0] + [1] \* (EXPONENT\_WIDTH - 1) + [0] + [1] \* MANTISS\_WIDTH

    print(f'{seeBin(num)} {seeNumber(num)} = максимальне додатнє представлення (2 - 2^(-{MANTISS\_WIDTH})) \* 2^{MAXIMUM\_EXP - 1}')

    num = [1] + [1] \* (EXPONENT\_WIDTH - 1) + [0] + [1] \* MANTISS\_WIDTH

    print(f'{seeBin(num)} {seeNumber(num)} = мінімальне від’ємне преставлення')

    num = numToBin(1.0)

    print(f'{seeBin(num)} {seeNumber(num)} = число +1,0Е0')

    num = [0] + [1] \* EXPONENT\_WIDTH + [0] \* MANTISS\_WIDTH

    print(f'{seeBin(num)} {seeNumber(num)} = значення +inf')  *# +∞*

    num = [1] + [1] \* EXPONENT\_WIDTH + [0] \* MANTISS\_WIDTH

    print(f'{seeBin(num)} {seeNumber(num)} = значення -inf')  *# -∞*

    num = [0] + [0] \* (EXPONENT\_WIDTH) + [0] \* (MANTISS\_WIDTH - 1) + [1]

    print(f'{seeBin(num)} {seeNumber(num)} = будь-який варіант для ненормалізованого ЧПТ')

    num = [0] + [0] \* (EXPONENT\_WIDTH + MANTISS\_WIDTH)

    print(f'{seeBin(num)} {seeNumber(num)} = +0.0')

    num = [1] + [0] \* (EXPONENT\_WIDTH + MANTISS\_WIDTH)

    print(f'{seeBin(num)} {seeNumber(num)} = -0.0')

    num = [0] + [1] \* (EXPONENT\_WIDTH) + [0] \* (MANTISS\_WIDTH - 1) + [1]

    print(f'{seeBin(num)} {seeNumber(num)} = будь-який варіант для NaN-значення')

*#  IEEE 754*

*# =======================*

def setExp(*arr*, *exp*):

*exp* += EXP\_SHIFT

    pos = EXPONENT\_WIDTH

*while* *exp*:

*arr*[pos] = *exp* % 2

        pos -= 1

*exp* = *exp* // 2

def isZero(*bin*):

    \_, exp, mant = binCut(*bin*)

*if* mant != 0:

*return* False

*if* exp != MINIMUM\_EXP:

*return* False

*return* True

def isInfinity(*bin*):

    \_, exp, mant = binCut(*bin*)

*if* mant != 0:

*return* False

*if* exp != MAXIMUM\_EXP:

*return* False

*return* True

def isNan(*bin*):

*if* isInfinity(*bin*):

*return* False

    exp = binCut(*bin*)[1]

*return* exp == MAXIMUM\_EXP

def isDenormalized(*bin*):

*if* isZero(*bin*):

*return* False

    exp = binCut(*bin*)[1]

*return* exp == MINIMUM\_EXP

*# bin array cutter*

def binCut(*bin*):

    sign = *bin*[0]

    exponent = 0

*for* i *in* range(1, EXPONENT\_WIDTH + 1):

        exponent \*= 2

        exponent += *bin*[i]

    exponent -= EXP\_SHIFT

    mantiss = 0

*for* i *in* range(EXPONENT\_WIDTH + 1, WIDTH):

        mantiss \*= 2

        mantiss += *bin*[i]

*return* sign, exponent, mantiss

*# bin array to float*

def seeNumber(*bin*):

    sign, exp, mant = binCut(*bin*)

*if* isZero(*bin*):

*if* sign == 1:

*return* ('-0.0')

*else*:

*return* ('+0.0')

*elif* isInfinity(*bin*):

*if* sign == 1:

*return* ('-inf')

*else*:

*return* ('+inf')

*elif* isNan(*bin*):

*return* ('NaN')

*elif* isDenormalized(*bin*):

*return* ((-1) \*\* sign) \* (2 \*\* exp) \* (mant / (2 \*\* MANTISS\_WIDTH))

*else*:

*return* ((-1) \*\* sign) \* (2 \*\* exp) \* (1 + mant / (2 \*\* MANTISS\_WIDTH))

*# bin array to string*

def seeBin(*bin*):

    arr = []

*# sign*

    arr.append(str(*bin*[0]))

*# exponenta*

    arr.append(''.join(map(str, *bin*[1:EXPONENT\_WIDTH + 1])))

*# mantissa*

    arr.append(''.join(map(str, *bin*[EXPONENT\_WIDTH + 1:])))

    mantiss\_forgotten\_bit = ''

*if* isZero(*bin*):

        mantiss\_forgotten\_bit = '0'

*elif* isInfinity(*bin*):

        mantiss\_forgotten\_bit = '1'

*elif* isNan(*bin*):

        mantiss\_forgotten\_bit = '0'

*else*:

*if* isDenormalized(*bin*):

            mantiss\_forgotten\_bit = '0'

*else*:

            mantiss\_forgotten\_bit = '1'

    arr.insert(2, mantiss\_forgotten\_bit)

*return* ' '.join(arr)

*# int to bin array*

def numToBin(*num*):

    bin = [0] \* WIDTH

    string = str(*num*)

*# sign*

    sign = 0

*if* string.startswith("-"):

        sign = 1

        string = string[1:]

*elif* string.startswith("+"):

        string = string[1:]

*# float split*

*if* '.' *in* string:

        a1, a2 = string.split('.')

*else*:

        a1, a2 = string, '0'

    div = 10 \*\* len(a2)

    a1, a2 = int(a1), int(a2)

*# integer part of number*

    binA1 = []

*while* a1:

        binA1.insert(0, a1 % 2)

        a1 = a1 // 2

*# +/- inf*

*if* len(binA1) > MAXIMUM\_EXP:  *# +/- inf*

        bin[0] = sign

        setExp(bin, MAXIMUM\_EXP)

*else*:

*# fractional part of a number*

        binA2 = []

*while* a2 *and* len(binA2) < EXP\_SHIFT + MANTISS\_WIDTH + 1:

            a2 \*= 2

            binA2.append(a2 // div)

            a2 %= div

*# integer number*

*if* binA1:

            bin[0] = sign

            exponent = len(binA1) - 1

            setExp(bin, exponent)

            temp = binA1[1:] + binA2

*for* i *in* range(min(len(temp), MANTISS\_WIDTH)):

                bin[1 + EXPONENT\_WIDTH + i] = temp[i]

*# float number*

*else*:

            exponent = -1

*for* i *in* range(len(binA2)):

*if* binA2[i] == 1:

                    exponent = i + 1

*break*

*# +/- 0.0*

*if* exponent == -1 *or* exponent > EXP\_SHIFT + MANTISS\_WIDTH:

                bin[0] = sign

*# denormalized*

*elif* exponent > EXP\_SHIFT:

                bin[0] = sign

*for* i *in* range(min(len(binA2) - EXP\_SHIFT, MANTISS\_WIDTH)):

                    bin[1 + EXPONENT\_WIDTH + i] = binA2[EXP\_SHIFT + i]

*# |number| < 1*

*else*:

                bin[0] = sign

                setExp(bin, -exponent)

*for* i *in* range(min(len(binA2) - exponent, MANTISS\_WIDTH)):

                    bin[1 + EXPONENT\_WIDTH + i] = binA2[exponent + i]

*return* bin

*# operations*

*# ===================*

def push(*num*, *b*=True):

*if* (len(stack) > STACK\_SIZE):

*raise* ValueError("Stack overflow")

*if* *b*:

        firstTact("Push", str(*num*))

        binNum = numToBin(*num*)

        stack.append(binNum)

        secondTact("Push", *num*)

*else*:

        binNum = numToBin(*num*)

        stack.append(binNum)

def pop(*operation*=" ", *b*=True):

*if* (len(stack) < 1):

*raise* ValueError(f"\nDurring: {*operation*}\nError: Stack empty")

*if* *b*:

        firstTact("Pop", str(num))

        num = stack.pop()

        secondTact("Pop", seeNumber(num))

*return* num

*else*:

*return* stack.pop()

def ln():

    firstTact("Ln")

    bin = pop("Ln", False)

    num = seeNumber(bin)

    push(log(num), False)

    secondTact("Ln", num)

def reverse():

    firstTact("Reverse")

    a = seeNumber(pop("Reverse", False))

    b = seeNumber(pop("Reverse", False))

    push(a, False)

    push(b, False)

    secondTact("Reverse", b)

def plus():

    firstTact("Plus")

    a = seeNumber(pop("Plus", False))

    b = seeNumber(pop("Plus", False))

    push(a+b, False)

    secondTact("Plus", b)

def sinn():

    firstTact("Sin")

    a = seeNumber(pop("Sin", False))

    push(sin(a), False)

    secondTact("Sin", a)

def coss():

    firstTact("Cos")

    a = seeNumber(pop("Cos", False))

    push(cos(a), False)

    secondTact("Cos", a)

def mult():

    firstTact("Mult")

    a = seeNumber(pop("Mult", False))

    b = seeNumber(pop("Mult", False))

    push(a\*b, False)

    secondTact("Mult", b)

def main(*x*=5, *y*=3):

    print("\nVariant: 8.10.11 \n")

    print("Do you want to see info? y|n")

    b = input("")

*if* b == "y" *or* b == "Y":

        seeInfo()

*# 8.10.11*

*# sin(ln(y) + 2) \* cos(x)*

    print("\nPlease enter the variables for sin(ln(y) + 2)\*cos(x)")

*x* = input("\nx: ")

*y* = input("y: ")

    print("\n")

*#         Formula iteration             Stack:*

    push(*x*)   *# x                           { x }*

    push(2)   *# 2                           { 2, x }*

    push(*y*)   *# y                           { y, 2, x }*

    ln()      *# ln(y)                       { ln(y), 2, x }*

    plus()    *# ln(y) + 2                   { ln(y) + 2, x }*

    sinn()    *# sin(ln(y) + 2)              { sin(ln(y) + 2), x }*

    reverse()  *# x                           { x, sin(ln(y) + 2) }*

    coss()    *# cos(x)                      { cos(x), sin(ln(y) + 2) }*

    mult()    *# sin(ln(y) + 2) \* cos(x)     { sin(ln(y) + 2) \* cos(x) }*

    ans = seeNumber(pop("Answer", False))

    print(f"\nAnswer = {ans}\n")

*return* ans

main()