СПЕЦІАЛЬНІ РОЗДІЛИ

ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ No1

Багаторозрядна арифметика

1. Мета роботи

Отримання практичних навичок програмної реалізації багаторозрядної арифметики; ознайомлення з прийомами ефективної реалізації критичних по часу ділянок програмного коду та методами оцінки їх ефективності.

2. Завдання до комп’ютерного практикуму

Реалізувати операції:

* Переведення констант у формат великого числа
* Додавання
* Віднімання
* Множення
* Піднесення до степеню
* Ділення, знаходженя остачі
* Визначення номеру старшого ненульового біта числа
* Бітові зсуви вправо та ліво

Перевірити коректність операцій та знайти визначити середній час виконання

3. Код :

from textwrap import wrap

class LongInt:

    hex\_charset = '0123456789abcdef'

    numeral\_system = 16

    length = 512

    def \_\_init\_\_(self, number):

        match number:

            case True:

                number = '1'

            case False:

                number = '0'

            case \_:

                number = number[-self.length:]

                for char in number.lower():

                    if char not in self.hex\_charset:

                        print('Invalid number format!')

                        number = ''

                        break

        self.number = '0' \* (self.length - len(number)) + number.lower()

    def to\_binary(self):

        result = ''

        number = self.number.lstrip('0')

        for i in number:

            if i == '0':

                result = result + '0000'

            elif i == '1':

                result = result + '0001'

            elif i == '2':

                result = result + '0010'

            elif i == '3':

                result = result + '0011'

            elif i == '4':

                result = result + '0100'

            elif i == '5':

                result = result + '0101'

            elif i == '6':

                result = result + '0110'

            elif i == '7':

                result = result + '0111'

            elif i == '8':

                result = result + '1000'

            elif i == '9':

                result = result + '1001'

            elif i == 'a':

                result = result + '1010'

            elif i == 'b':

                result = result + '1011'

            elif i == 'c':

                result = result + '1100'

            elif i == 'd':

                result = result + '1101'

            elif i == 'e':

                result = result + '1110'

            elif i == 'f':

                result = result + '1111'

        return '0' \* (self.length\*4 - len(result)) + result

    def from\_binary(self, binary):

        result = ''

        result\_arr = tuple(wrap(binary, 4))

        for binary in result\_arr:

            if binary == '0000':

                result = result + '0'

            elif binary == '0001':

                result = result+'1'

            elif binary == '0010':

                result = result + '2'

            elif binary == '0011':

                result = result + '3'

            elif binary == '0100':

                result = result + '4'

            elif binary == '0101':

                result = result + '5'

            elif binary == '0110':

                result = result + '6'

            elif binary == '0111':

                result = result + '7'

            elif binary == '1000':

                result = result + '8'

            elif binary == '1001':

                result = result + '9'

            elif binary == '1010':

                result = result + 'a'

            elif binary == '1011':

                result = result + 'b'

            elif binary == '1100':

                result = result + 'c'

            elif binary == '1101':

                result = result + 'd'

            elif binary == '1110':

                result = result + 'e'

            elif binary == '1111':

                result = result + 'f'

        return result

    def lshift(self, value=1):

        binary = self.to\_binary()

        return LongInt(self.from\_binary(binary[value:] + '0' \* value))

    def rshift(self, value=1):

        binary = self.to\_binary()

        return LongInt(self.from\_binary('0' \* value + binary[:-value]))

    def elder\_bit(self):

        binary = self.to\_binary().lstrip('0')

        return len(binary)-1

    def add(self, value):

        result = ''

        next\_add = False

        for i in range(self.length - 1, -1, -1):

            single\_add = self.hex\_charset.index(self.number[i]) + self.hex\_charset.index(value.number[i]) + next\_add

            if single\_add >= self.numeral\_system:

                next\_add = True

                single\_add = single\_add % self.numeral\_system

            else:

                next\_add = False

            result = self.hex\_charset[single\_add] + result

        return LongInt(result)

    def lt(self, value):

        for i in range(self.length):

            if self.hex\_charset.index(self.number[i]) > self.hex\_charset.index(value.number[i]):

                return False

            elif self.hex\_charset.index(self.number[i]) < self.hex\_charset.index(value.number[i]):

                return True

        return True

    def sub(self, value):

        if self.lt(value):

            return LongInt('')

        result = ''

        next\_sub = False

        for i in range(self.length - 1, -1, -1):

            single\_sub = self.hex\_charset.index(self.number[i]) - self.hex\_charset.index(value.number[i]) - next\_sub

            if single\_sub < 0:

                next\_sub = True

                single\_sub = single\_sub + self.numeral\_system

            else:

                next\_sub = False

            result = self.hex\_charset[single\_sub] + result

        return LongInt(result)

    def mul(self, value):

        binary = value.to\_binary()

        result = LongInt('')

        for i in range(len(binary)):

            if binary[i] == '1':

                result = result.add(self.lshift(len(binary)-i-1))

        return result

    def div(self, value):

        if str(value) == '0':

            return None

        if self.lt(value):

            return LongInt(''), self

        shifts = self.elder\_bit() - value.elder\_bit()

        rem = LongInt(self.number)

        divider = value.lshift(shifts)

        result = ''

        for \_ in range(shifts+1):

            if rem.lt(divider):

                result = result + '0'

            else:

                result = result + '1'

                rem = rem.sub(divider)

            divider = divider.rshift()

        result = LongInt(self.from\_binary('0'\*(2048-len(result)) + result))

        return result, self.sub(result.mul(value))

    def pow(self, value):

        value = int(value.number, base=16)

        if value < 0:

            return None

        if value == 0:

            return LongInt('1')

        if value == 1:

            return self

        result = LongInt('1')

        binary = bin(value)[2:]

        for i in range(len(binary)):

            result = result.mul(result)

            if binary[i] == '1':

                result = result.mul(self)

        return result

    def \_\_repr\_\_(self):

        repr = self.number.lstrip('0')

        if not repr:

            repr = '0'

        return repr

4. Перевірка на коректність:

for i in range(100):

    a = LongInt(''.join(choices('0123456789abcdef', k=512)))

    b = LongInt(''.join(choices('0123456789abcdef', k=512)))

    c = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num1 = a.add(b).mul(c)  # (a+b)\*c

    num2 = c.mul(a.add(b))  # c\*(a+b)

    num3 = a.mul(c).add(b.mul(c))  # a\*c+b\*c

    if str(num1) != str(num2) != str(num3):

        print('ERROR!')

        break

x = LongInt(''.join(choices('0123456789abcdef', k=512)))

result = x

for i in range(300):

    result = result.add(x)

if str(result) != str(x.mul(LongInt('12d'))):

    print('ERROR!')

Перевірка проходить без помилок

5. Визначення часу роботи:

start\_init = time()

for \_ in range(1000):

    num = LongInt(''.join(choices('0123456789abcdef', k=512)))

end\_init = time()

start\_to\_binary = time()

for \_ in range(1000):

    num = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num.to\_binary()

end\_to\_binary = time()

start\_from\_binary = time()

for \_ in range(1000):

    num = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num.from\_binary(num.to\_binary())

end\_from\_binary = time()

start\_lshift = time()

for \_ in range(1000):

    num = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num.lshift()

end\_lshift = time()

start\_rshift = time()

for \_ in range(1000):

    num = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num.rshift()

end\_rshift = time()

start\_elder\_bit = time()

for \_ in range(1000):

    num = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num.elder\_bit()

end\_elder\_bit = time()

start\_add = time()

for \_ in range(1000):

    num1 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num2 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num1.add(num2)

end\_add = time()

start\_sub = time()

for \_ in range(1000):

    num1 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num2 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num1.sub(num2)

end\_sub = time()

start\_lt = time()

for \_ in range(1000):

    num1 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num2 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num1.lt(num2)

end\_lt = time()

start\_mul = time()

for \_ in range(10):

    num1 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num2 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num1.mul(num2)

end\_mul = time()

start\_div = time()

for \_ in range(10):

    num1 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num2 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num1.div(num2)

end\_div = time()

start\_pow = time()

for \_ in range(10):

    num1 = LongInt(''.join(choices('0123456789abcdef', k=512)))

    num2 = LongInt(''.join(choices('0123456789abcdef', k=3)))

    num1.pow(num2)

end\_pow = time()

init = end\_init-start\_init

to\_bin = end\_to\_binary - start\_to\_binary - init

from\_bin = end\_from\_binary - start\_from\_binary - init - to\_bin

lshift = end\_lshift - start\_lshift - init

rshift = end\_rshift - start\_rshift - init

elder\_bit = end\_elder\_bit - start\_elder\_bit - init

add = end\_add - start\_add-init\*2

sub = end\_sub - start\_sub - init\*2

lt = end\_lt - start\_lt - init\*2

mul = (end\_mul-start\_mul-init\*2)\*100

div = (end\_div-start\_div-init\*2)\*100

pow = (end\_pow-start\_pow-init\*2)\*100

Таблиця з отриманими результатами (час у мілісекундах):

Час виконання піднесення до степеню зазначаю окремо, так як порівнювати з іншими операціями його не має сенсу – 20507мс = 20,5с



