Київський національний університет

імені Тараса Шевченка

Звіт

до лабораторної роботи №3 на тему:

*“Представлення даних у форматі чисел з плаваючою*

*точкою”*

*Студентки другого курсу*

*Групи К-23*

*Факультету комп’ютерних наук*

*та кібернетики*

*Старостюк Дарії*

Київ-2023

# Мета

Метою лабораторної роботи є розробка програмної моделі співпроцесора, реалізація його імітаційної моделі та розробка представлення введеного у діалозі десяткового числа у експоненціальній формі у форматі ЧПТ

# Постановка задачі

Розробити програму представлення введеного у діалозі десяткового числа в експоненціальній формі у визначеного варіантом форматі ЧПТ.

В індивідуальному варіанті буде визначена конкретна**:**

1)  Обов’язкова для реалізації формула для обрахунку співпроцесором

2) Індивідуальний варіант  для представлення ЧПТ вигляду ЦЦ*x*.ЦЦ*m*, де ЦЦ*x* задає розмір характеристики у бітах та ЦЦ*m* задає кількість біт у мантисі без врахування неявного біта.

Варіант 8 h 35 m

F(a,b)=a-b+2.5/a-2.5/b– формула для обрахунку співпроцесором

# Опис роботи

* Імітований співпроцесор має 8 регістрів (об’єднані в стек);
* Доступ до регістрів здійснюється за допомогою індексів
* Формат числа з плаваючою точкою (ЧПТ) має стандартну структуру: sHM де: s –біт знаку мантиси, H – 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;
* Дані в регістрах подаються в форматі IEEE 754;
* Регістр команд, регістр тактів та регістр стану(фіксується знак результату виконання команди);
* Обрахунки заданої формули проводяться потактово: 1-й - занесення поточної команди у регістр команди; 2-й - виконання команди
* Для обрахунку формули використовуються два заданих користувачем на початку роботи значення (a та b)
* Результат зберігається в першому регістрі
* Також опрацьовуються і такі значення:
  + мінімальне за абсолютною величиною ненульове представлення;
  + максимальне додатне представлення;
  + мінімальне від’ємне представлення;
  + число +1,0Е0;
  + значення +∞;
  + значення -∞;
  + NaN-значення

# Приклад роботи програми

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Код програми

h = 8  
m = 35  
  
class Processor:  
 def \_\_init\_\_(self):  
 self.characteristic\_bits\_amount = h  
 self.characteristic\_bits\_bias = (2 \*\* h // 2) - 1  
 self.mantissa\_bits\_amount = m  
  
 self.register\_1 = None  
 self.register\_2 = None  
 self.register\_3 = None  
 self.register\_4 = None  
 self.register\_5 = None  
 self.register\_6 = None  
 self.register\_7 = None  
 self.register\_8 = None  
  
 self.max\_bound = 2 \*\*(2\*\*(h-1))  
 self.min\_bound = -2 \*\*(2\*\*(h-1))  
  
 @staticmethod  
 def print\_to\_console(s, h, m, num):  
 return str(s) + ' ' + str(h) + ' ' + str(m) + ' Represents: ' + str(num)  
  
 @staticmethod  
 def invert\_bits(num):  
 inv\_n = ''  
 for n in num:  
 if n == '1':  
 inv\_n = inv\_n + '0'  
 if n == '0':  
 inv\_n = inv\_n + '1'  
 return inv\_n  
  
 @staticmethod  
 def decimal\_converter(nu):  
 while nu >= 1:  
 nu /= 10.0  
 return nu  
  
 def dec\_to\_ieee754(self, number):  
 if number is None:  
 return None, '', ''  
 if number > self.max\_bound:  
 raise Exception(f'Too positive number {number} for upper bound with h {self.characteristic\_bits\_amount}:'  
 f' {self.max\_bound} 2^(-1)\*2\*\*{self.characteristic\_bits\_amount} - 1')  
 if number < self.min\_bound:  
 raise Exception(f'Too negative number {number} for upper bound with h {self.characteristic\_bits\_amount}:'  
 f' {self.max\_bound} 2^(-1)\*2\*\*{self.characteristic\_bits\_amount}')  
 # convert 1E-7 form or 1E+10form types  
 number = str(number)  
 if number[1] == 'e':  
 if number[2] == '-':  
 n = '0.' + '0' \* (int(number[3:]) - 1) + '1'  
 if number[0] == '-':  
 number = '-' + n  
 else:  
 number = n  
 if number[2] == '+':  
 n = '1' + '0' \* (int(number[3:])) + '.0'  
 if number[0] == '-':  
 number = '-' + n  
 elif number[0] == '+':  
 number = n  
 whole, dec = number.split(".")  
 h = 0  
 whole = int(whole)  
 if whole < 1:  
 for i, dig in enumerate(dec):  
 if dig != '0':  
 h = - i  
 break  
 dec = int(dec)  
 sign = '0'  
 if whole < 0:  
 sign = '1'  
 whole = abs(whole)  
 res = bin(whole).lstrip("0b")  
 sc\_res = res  
 if whole > 1:  
 h\_ = len(res) - 1  
 else:  
 h\_ = h  
  
  
 def decimal\_converter(nu):  
 while nu >= 1:  
 nu /= 10  
 return nu  
  
 for x in range(self.mantissa\_bits\_amount):  
 wd = str(float(decimal\_converter(int(dec)) \* 2))  
 if '.' in wd:  
 whole, dec = wd.split(".")  
 else:  
 whole = int(wd)  
 dec = '0'  
 dec = int(dec)  
 sc\_res = sc\_res + str(whole)  
 ch = bin(self.characteristic\_bits\_bias + h\_).lstrip('0b')  
 ch = '0' \* (self.characteristic\_bits\_amount - len(ch)) + ch  
 if sign == '0':  
 num = int(sc\_res, 2) \* 2 \*\* (h - self.mantissa\_bits\_amount)  
 return sign, ch, sc\_res[:self.mantissa\_bits\_amount], num, h  
 if sign == '1':  
 num = - int(sc\_res, 2) \* 2 \*\* (h - self.mantissa\_bits\_amount)  
 return sign, ch, sc\_res[:self.mantissa\_bits\_amount], num, h  
 else:  
 return 0, '0' \* self.characteristic\_bits\_amount, '0' \* (self.mantissa\_bits\_amount - 1) + '1', 0, 0  
  
 def mov(self, x):  
 if x is not None:  
 x = float(x)  
 if self.min\_bound < x < self.max\_bound:  
 self.register\_8 = self.register\_7  
 self.register\_7 = self.register\_6  
 self.register\_6 = self.register\_5  
 self.register\_5 = self.register\_4  
 self.register\_4 = self.register\_3  
 self.register\_3 = self.register\_2  
 self.register\_2 = self.register\_1  
 self.register\_1 = x  
 else:  
 if x > self.max\_bound:  
 self.register\_1 = float('inf')  
 elif x < self.min\_bound:  
 self.register\_1 = float('-inf')  
 else:  
 return None  
  
 def dubl(self):  
 self.register\_8 = self.register\_7  
 self.register\_7 = self.register\_6  
 self.register\_6 = self.register\_5  
 self.register\_5 = self.register\_4  
 self.register\_4 = self.register\_3  
 self.register\_3 = self.register\_2  
 self.register\_2 = self.register\_1  
  
 def swap(self):  
 temp = self.register\_1  
 self.register\_1 = self.register\_2  
 self.register\_2 = temp  
  
 def add(self):  
 nw = self.register\_1 + self.register\_2  
 nw\_bit = self.dec\_to\_ieee754(nw)[3]  
  
 if (self.register\_1 == float("inf") and self.register\_2 == float("-inf")) or \  
 (self.register\_1 == float("-inf") and self.register\_2 == float("inf")):  
 self.register\_1 = float("nan")  
 elif self.register\_1 == float("inf") or self.register\_2 == float("inf"):  
 self.register\_1 = float("inf")  
 elif self.register\_1 == float("-inf") or self.register\_2 == float("-inf"):  
 self.register\_1 = float("-inf")  
 else:  
 if self.min\_bound < nw < self.max\_bound:  
 self.register\_1 = nw  
 self.register\_2 = self.register\_3  
 self.register\_3 = self.register\_4  
 self.register\_4 = self.register\_5  
 self.register\_5 = self.register\_6  
 self.register\_6 = self.register\_7  
 self.register\_7 = self.register\_8  
 self.register\_8 = None  
 else:  
 raise Exception(f'{self.register\_1} + {self.register\_2} = {nw} value not in bounds: '  
 f'{self.min\_bound}, {self.max\_bound}')  
 if self.register\_1 == float("inf") or self.register\_1 == float("-inf") or self.register\_1 == float("nan"):  
 print(f"Result: {self.register\_1}")  
 return  
  
 def sub(self):  
 nw = self.register\_1 - self.register\_2  
 nw = self.dec\_to\_ieee754(nw)[3]  
 if self.min\_bound < nw < self.max\_bound:  
 self.register\_1 = self.dec\_to\_ieee754(nw)[3]  
 self.register\_2 = self.register\_3  
 self.register\_3 = self.register\_4  
 self.register\_4 = self.register\_5  
 self.register\_5 = self.register\_6  
 self.register\_6 = self.register\_7  
 self.register\_7 = self.register\_8  
 self.register\_8 = None  
 else:  
 raise Exception(f'{self.register\_1} - {self.register\_2} = {nw} value not in bounds:'  
 f'{self.min\_bound}, {self.max\_bound}')  
  
 def mult(self):  
 nw = self.register\_1 \* self.register\_2  
 nw = self.dec\_to\_ieee754(nw)[3]  
 if self.min\_bound < nw < self.max\_bound:  
 self.register\_1 = self.dec\_to\_ieee754(nw)[3]  
 self.register\_2 = self.register\_3  
 self.register\_3 = self.register\_4  
 self.register\_4 = self.register\_5  
 self.register\_5 = self.register\_6  
 self.register\_6 = self.register\_7  
 self.register\_7 = self.register\_8  
 self.register\_8 = None  
 else:  
 raise Exception(f'{self.register\_1} \* {self.register\_2} = {nw} value not in bounds:'  
 f'{self.min\_bound}, {self.max\_bound}')  
  
 def div(self):  
 if self.register\_2 is not None and self.register\_2 != 0:  
 nw = self.register\_1 / self.register\_2  
 if nw == float("inf") or nw == float("-inf") or nw > self.max\_bound or nw < self.min\_bound:  
 self.register\_1 = float("inf") if nw > 0 else float("-inf")  
 else:  
 self.register\_1 = nw  
 self.register\_2 = self.register\_3  
 self.register\_3 = self.register\_4  
 self.register\_4 = self.register\_5  
 self.register\_5 = self.register\_6  
 self.register\_6 = self.register\_7  
 self.register\_7 = self.register\_8  
 self.register\_8 = None  
 else:  
 self.register\_1 = float("inf") if self.register\_1 > 0 else float("-inf")  
 if self.register\_1 == float("inf") or self.register\_1 == float("-inf"):  
 print(f"Result: {self.register\_1}")  
 return  
  
 def ex\_com(self, current\_command, cl):  
 print(f"Command № {cl // 2 + 1} clock rate {cl + 1}\n")  
 parts = [p for p in current\_command.split()]  
 current\_command = parts[0]  
 register\_1 = self.dec\_to\_ieee754(self.register\_1)  
 register\_2 = self.dec\_to\_ieee754(self.register\_2)  
 register\_3 = self.dec\_to\_ieee754(self.register\_3)  
 register\_4 = self.dec\_to\_ieee754(self.register\_4)  
 register\_5 = self.dec\_to\_ieee754(self.register\_5)  
 register\_6 = self.dec\_to\_ieee754(self.register\_6)  
 register\_7 = self.dec\_to\_ieee754(self.register\_7)  
 register\_8 = self.dec\_to\_ieee754(self.register\_8)  
 print(f"1R: {register\_1[0]} {register\_1[1]} {register\_1[2]} {self.register\_1}\n"  
 f"2R: {register\_2[0]} {register\_2[1]} {register\_2[2]} {self.register\_2}\n"  
 f"3R: {register\_3[0]} {register\_3[1]} {register\_3[2]} {self.register\_3}\n"  
 f"4R: {register\_4[0]} {register\_4[1]} {register\_4[2]} {self.register\_4}\n"  
 f"5R: {register\_5[0]} {register\_5[1]} {register\_5[2]} {self.register\_5}\n"  
 f"6R: {register\_6[0]} {register\_6[1]} {register\_6[2]} {self.register\_6}\n"  
 f"7R: {register\_7[0]} {register\_7[1]} {register\_7[2]} {self.register\_7}\n"  
 f"8R: {register\_8[0]} {register\_8[1]} {register\_8[2]} {self.register\_8}\n"  
 '\n'  
 f"IR: {current\_command}")  
 print()  
 if current\_command == 'load':  
 if len(parts) != 2:  
 raise Exception(f'PC {cl // 2 + 1} TC {cl + 1}')  
 value = parts[1]  
 self.mov(x=value)  
 elif current\_command == 'dubl':  
 self.dubl()  
 elif current\_command == 'swap':  
 self.swap ()  
 elif current\_command == 'add':  
 self.add()  
 elif current\_command == 'sub':  
 self.sub()  
 elif current\_command == 'mult':  
 self.mult()  
 elif current\_command == 'div':  
 self.div()  
 else:  
 raise Exception(f'Unknown command {parts[0]}')  
 if self.register\_1 in [float('inf'), -float('inf')]:  
 return True  
 print(f"Command № {cl // 2 + 1} clock rate {cl + 2}\n")  
 register\_1 = self.dec\_to\_ieee754(self.register\_1)  
 register\_2 = self.dec\_to\_ieee754(self.register\_2)  
 register\_3 = self.dec\_to\_ieee754(self.register\_3)  
 register\_4 = self.dec\_to\_ieee754(self.register\_4)  
 register\_5 = self.dec\_to\_ieee754(self.register\_5)  
 register\_6 = self.dec\_to\_ieee754(self.register\_6)  
 register\_7 = self.dec\_to\_ieee754(self.register\_7)  
 register\_8 = self.dec\_to\_ieee754(self.register\_8)  
 print(f"1R: {register\_1[0]} {register\_1[1]} {register\_1[2]} {self.register\_1}\n"  
 f"2R: {register\_2[0]} {register\_2[1]} {register\_2[2]} {self.register\_2}\n"  
 f"3R: {register\_3[0]} {register\_3[1]} {register\_3[2]} {self.register\_3}\n"  
 f"4R: {register\_4[0]} {register\_4[1]} {register\_4[2]} {self.register\_4}\n"  
 f"5R: {register\_5[0]} {register\_5[1]} {register\_5[2]} {self.register\_5}\n"  
 f"6R: {register\_6[0]} {register\_6[1]} {register\_6[2]} {self.register\_6}\n"  
 f"7R: {register\_7[0]} {register\_7[1]} {register\_7[2]} {self.register\_7}\n"  
 f"8R: {register\_8[0]} {register\_8[1]} {register\_8[2]} {self.register\_8}\n"  
 '\n'  
 f"IR: {current\_command}")  
 print()  
  
 def run(self, commands):  
 result\_register = None  
 computation\_stopped = False  
 for c, command in enumerate(commands):  
 print('\_' \* 50)  
 stop\_computation = self.ex\_com(current\_command=command, cl=2 \* c)  
 if stop\_computation:  
 result\_register = getattr(self, f"register\_1")  
 computation\_stopped = True  
 break  
 result\_register = getattr(self, f"register\_1")  
 print('\_' \* 50)  
 if computation\_stopped:  
 if result\_register == float('inf'):  
 print('1R:0 11111111 000000000000000000000000000000000000 inf')  
 elif result\_register == -float('inf'):  
 print('1R:1 11111111 000000000000000000000000000000000000 -inf')  
 else:  
 print('Result:')  
 if result\_register is not None:  
 if result\_register == float('inf'):  
 print('1R:0 11111111 000000000000000000000000000000000000 inf')  
 elif result\_register == -float('inf'):  
 print('1R:1 11111111 000000000000000000000000000000000000 -inf')  
 else:  
 sign, ch, sc\_res, num, h = self.dec\_to\_ieee754(result\_register)  
 print(f'1R: {sign} {ch} {sc\_res} {result\_register}')  
 input()  
  
def main():  
 a = input("Введіть значення 'a': ")  
 b = input("Введіть значення 'b': ")  
 print("\nF=a-b+2.5/a-2.5/b")  
  
 commands = [  
 f'load {b}',  
 f'load {a}',  
 'load 2.5',  
 'div',  
 'swap',  
 'load 2.5',  
 'div',  
 'sub',  
 f'load {b}',  
 f'load {a}',  
 'sub',  
 'add'  
 ]  
  
 proc = Processor()  
 proc.run(commands)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# 