Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра системного аналізу та теорії прийняття рішень

Звіт

з лабораторної роботи № 2

на тему:

**«Імітаційна модель процесора»**

Варіант 1, 4, 14

Студентки другого курсу

групи К-23

Старостюк Дарії Олегівни

Факультету комп’ютерних наук

та кібернетики

Київ – 2023

# Постановка задачі

Розробити програмну модель процесора та реалізувати його імітаційну (тобто комп’ютерну) модель.

Має бути реалізовано:

1)    розміщення інтерпретуємої програми у текстовому файлі (наприклад, один рядок=одна команда);

2)    мінімум 2 команди (одна з них - занесення значення у регістр, інші задаються варіантом);

3)    для операндів/регістрів представлення побітно, можливо, для деяких варіантів із побайтним групуванням бітів;

4)    фіксація у регістрі стану  як мінімум знаку результату виконання команди;

5)    потактове виконання команд (наприклад, 1-й такт – занесення команди у регістр команди, 2-й такт -  виконання операції і занесення результату).

# Індивідуальний варіант

1)    1-адресний

2)   оперативна пам’ять з 10 22-бітних полів

3)    Операнд з даними складається із байтів-символів із цифр та латиниці. Команда

перетворює набір символів у 1-му операнді у верхній/нижній регістр в залежності від

значення 2-го операнда команди (його значення вибрати самостійно), представленого

у:

         команді безпосередньо чи регістрі для безстекової реалізації;

         верхівці стека в стековій реалізації розміщення операндів.

Біти регістра/стека відображати із побайтовою розбивкою.

# Опис роботи

Імітований процесор має наступні регістри:

IR - регістр команди (містить текстовий запис команди, яка є поточною);

Rn - регістр;

PS - регістр статусу

PC - регістр лічильника команд;

TC - регістр лічильника тактів;

та команди:

mov - Копіює значення акумулятора в обраний регістр або на обраний регістр.

Приклади:

mov R1 - копіює значення акумулятора в регістр R1.

mov -5 - копіює від'ємне ціле число в акумулятор.

add - Додає значення з обраного регістру до акумулятора та зберігає результат в акумуляторі.

str upper - перетворює символьні дані в верхній регістр.

str lower - перетворює символьні дані в нижній регістр.

exit - Завершує роботу програми.

## Текстовий файл з командами:

mov 15  
mov R1  
mov -3  
mov R2  
add R1  
mov R3  
mov 65  
str lower  
exit

## Результати:

# CPU State:

# IR = mov 15

# PC = 1 TC = 1 PS = 0

# Acc = 00000000 00000000 000000

# R1 = 00000000 00000000 000000

# R2 = 00000000 00000000 000000

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov 15

# PC = 1 TC = 2 PS = 0

# Acc = 00000000 00000000 001111

# R1 = 00000000 00000000 000000

# R2 = 00000000 00000000 000000

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov R1

# PC = 2 TC = 1 PS = 0

# Acc = 00000000 00000000 001111

# R1 = 00000000 00000000 000000

# R2 = 00000000 00000000 000000

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov R1

# PC = 2 TC = 2 PS = 0

# Acc = 00000000 00000000 001111

# R1 = 00000000 00000000 001111

# R2 = 00000000 00000000 000000

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov -3

# PC = 3 TC = 1 PS = 0

# Acc = 00000000 00000000 001111

# R1 = 00000000 00000000 001111

# R2 = 00000000 00000000 000000

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov -3

# PC = 3 TC = 2 PS = 0

# Acc = 11111111 11111111 111101

# R1 = 00000000 00000000 001111

# R2 = 00000000 00000000 000000

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov R2

# PC = 4 TC = 1 PS = 0

# Acc = 11111111 11111111 111101

# R1 = 00000000 00000000 001111

# R2 = 00000000 00000000 000000

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov R2

# PC = 4 TC = 2 PS = 0

# Acc = 11111111 11111111 111101

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = add R1

# PC = 5 TC = 1 PS = 0

# Acc = 11111111 11111111 111101

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = add R1

# PC = 5 TC = 2 PS = 0

# Acc = 00000000 00000000 001100

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov R3

# PC = 6 TC = 1 PS = 0

# Acc = 00000000 00000000 001100

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 000000

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov R3

# PC = 6 TC = 2 PS = 0

# Acc = 00000000 00000000 001100

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 001100

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov 65

# PC = 7 TC = 1 PS = 0

# Acc = 00000000 00000000 001100

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 001100

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = mov 65

# PC = 7 TC = 2 PS = 0

# Acc = 00000000 00000001 000001

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 001100

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = str lower

# PC = 8 TC = 1 PS = 0

# Acc = 00000000 00000001 000001

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 001100

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# CPU State:

# IR = str lower

# PC = 8 TC = 2 PS = 0

# Acc = 00000000 00000001 100001

# R1 = 00000000 00000000 001111

# R2 = 11111111 11111111 111101

# R3 = 00000000 00000000 001100

# R4 = 00000000 00000000 000000

# R5 = 00000000 00000000 000000

# R6 = 00000000 00000000 000000

# R7 = 00000000 00000000 000000

# R8 = 00000000 00000000 000000

# R9 = 00000000 00000000 000000

# R10 = 00000000 00000000 000000

# ..............................

# Process finished with exit code 0Код програми

class CPU:  
 def \_\_init\_\_(self):  
 self.registers = {  
 'R1': [0] \* 22,  
 'R2': [0] \* 22,  
 'R3': [0] \* 22,  
 'R4': [0] \* 22,  
 'R5': [0] \* 22,  
 'R6': [0] \* 22,  
 'R7': [0] \* 22,  
 'R8': [0] \* 22,  
 'R9': [0] \* 22,  
 'R10': [0] \* 22  
 }  
 self.PC = 1  
 self.TC = 0  
 self.PS = 0  
 self.status\_register = [0]  
 self.accumulator = [0] \* 22  
 self.memory = [0] \* (2 \*\* 22)  
 self.IR = ''  
  
 self.print\_state()  
  
 def print\_state(self):  
 if self.IR:  
 print("CPU State:")  
 print(f"IR = {self.IR}")  
 print(f"PC = {self.PC} TC = {self.TC} PS = {self.PS}")  
 print(f"Acc = {self.bits\_byte\_format(self.accumulator)}")  
 for register, value in self.registers.items():  
 print(f"{register} = {self.bits\_byte\_format(value)}")  
 print("." \* 30)  
  
 def bits\_byte\_format(self, bits):  
 return ' '.join([''.join(map(str, bits[i:i + 8])) for i in range(0, 22, 8)])  
  
 def check\_overflow(self, bits):  
 max\_value = 2 \*\* 22 - 1  
 value = 0  
 for bit in bits:  
 value = (value << 1) | bit  
  
 if value > max\_value:  
 self.PS = 1  
 else:  
 self.PS = 0  
  
 def execute\_instruction\_tact\_by\_tact(self, raw\_command):  
 try:  
 split\_command = raw\_command.split()  
 command = split\_command[0]  
 self.IR = raw\_command  
 except (ValueError, IndexError):  
 print("Invalid command format")  
 return  
 self.TC = 1  
 self.print\_state()  
  
 if command == 'mov':  
 if len(split\_command) == 2:  
 operand = split\_command[1]  
  
 if operand.isdigit() or (operand[0] == '-' and operand[1:].isdigit()):  
 operand = int(operand)  
 operand\_binary = bin(operand & 0x3FFFFF)[2:].zfill(22)  
 operand = [int(bit) for bit in operand\_binary]  
  
 self.accumulator = operand  
 self.check\_overflow(operand)  
 elif operand in self.registers:  
 self.registers[operand] = self.accumulator  
 else:  
 print("Invalid operand or register name format")  
 self.PS = 1  
 else:  
 print("Invalid number of operands for mov command")  
 self.PS = 1  
  
  
 elif command == 'add':  
 if len(split\_command) == 2:  
 source\_register = split\_command[1]  
  
 if source\_register in self.registers:  
 carry = 0  
 temp\_accumulator = self.accumulator.copy()  
 for i in range(21, -1, -1):  
 sum\_bits = self.accumulator[i] + self.registers[source\_register][i] + carry  
 carry = sum\_bits // 2  
 temp\_accumulator[i] = sum\_bits % 2  
 self.status\_register[0] = carry  
 self.accumulator = temp\_accumulator  
 self.check\_overflow(temp\_accumulator)  
 else:  
 print(f"Invalid register name: {source\_register}")  
 self.PS = 1  
 else:  
 print("Invalid number of operands for add command")  
 self.PS = 1  
  
  
 elif command == 'str':  
  
 if len(split\_command) == 2:  
  
 destination\_location = split\_command[1]  
  
 if destination\_location.lower() == "upper" or destination\_location.lower() == "lower":  
  
 is\_upper = destination\_location.lower() == "upper"  
  
 operand\_data = int(''.join(map(str, self.accumulator)), 2)  
  
 if (65 <= operand\_data <= 90 or 97 <= operand\_data <= 122):  
  
 if (is\_upper and 97 <= operand\_data <= 122) or (not is\_upper and 65 <= operand\_data <= 90):  
 operand\_data = operand\_data - 32 if is\_upper else operand\_data + 32  
  
 else:  
  
 print("Invalid value for the second operand in the str command")  
  
 self.PS = 1  
  
 operand\_data = 0 # Анулюємо аккумулятор  
  
 operand\_bits = list(f'{operand\_data:08b}')  
  
 operand\_bits = [int(bit) for bit in operand\_bits]  
  
 operand\_bits = [0] \* (22 - len(operand\_bits)) + operand\_bits  
  
 self.accumulator = operand\_bits  
  
 self.check\_overflow(operand\_bits)  
  
 else:  
  
 print("Invalid value for the second operand in the str command")  
  
 self.PS = 1  
  
 else:  
  
 print("Invalid number of operands for str command")  
  
 self.PS = 1  
  
 self.check\_overflow(self.accumulator)  
  
 for register in self.registers.values():  
 self.check\_overflow(register)  
  
 self.TC = 2  
  
 self.print\_state()  
  
 self.PC += 1  
  
 def execute\_from\_file(self, file\_path):  
 with open(file\_path, 'r') as file:  
 commands = file.readlines()  
 for command in commands:  
 command = command.strip()  
 if command == 'exit':  
 break  
 else:  
 self.execute\_instruction\_tact\_by\_tact(command)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 cpu = CPU()  
  
 file\_path = "commands.txt"  
 cpu.execute\_from\_file(file\_path)