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 2, 7, 19

Студента другого курсу

групи К-23(2)

Тарасюка Олександра

Факультету комп’ютерних наук

та кібернетики

Київ – 2022

# Постановка задачі

## Задача

Необхідно 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/стек/ОП, інші задаються варіантом);
3. для операндів/регістрів представлення побітно, можливо, для деяких варіантів із побайтним групуванням бітів. Оперативна пам’ять має представлятися у 16-річному форматі;
4. фіксація у регістрі стану як мінімум знаку результату виконання команди;
5. потактове виконання команд (наприклад, 1-й такт – занесення команди у регстр команди, 2-й такт – інтерпретація операндів, 3-й такт – виконання операції і занесення результату).

## Задачі варіанту

Індивідуальний варіант завдання: 2, 7, 19.

1. 2-адресний процесор. Результат розмішується у 1-му операнді
2. Бітність регістрів/стеку та операндів команд: 24 біт;
3. Індивідуальна команда процесора: Логічний зсув вліво та вправо групи регістрів

## Рекомендації щодо виконання роботи

1. Щодо вибору кількості тактів для команд - мінімально достатньо двох: 1-й) занесення поточної команди у регістр команди; 2-й) виконання команди.
2. Щодо реалізації ОП - можна обмежитися роботою лише з 10-20 байтами (але вони мають бути відображеними на екрані).
3. Для відрахунку тактів найзручніше брати нажимання певної чи будь-якої клавіші клавіатури. Доцільно завжди мати можливість вийти із стану інтерпретації програми імітуємим процесором за допомогою, наприклад, Esc. Не варто реалізовувати програмної затримки для кожного такта у порівнянні із "тактуванням" за допомогою миші чи клавіатури, бо вибраний вами темп може перешкоджати аналізу виконуваних дій.
4. Файл інтерпретуємої програми має бути заданим у програмі або один раз у командному рядку запуску імітаційної моделі. Тобто програма не повинна пропонувати у діалозі вказати файл команд для виконання імітуємим процесором.
5. На екрані (достатньо в режимі скролінгу тексту і без рамок для даних у регістрах) для кожного такту повинні бути представлені такі структурні елементи процесору із даними в них:



# Реалізація

## Регістри та пам’ять

Модель використовує 4 регістри стану виконання програми та 1 регістр-акумулятор. Кожен з них містить 15 біт інформації.

* **R1-R4** – регістри (відображаються у двійковому вигляді);
* **PC** – лічильник команд (порядковий номер рядка команди) (відображається у десятковому вигляді);
* **TC** – лічильник тактів (відображається у десятковому вигляді);
* **PS** – регістр стану (відображається у двійковому вигляді).

Крім цього було реалізована оперативна пам’ять, яку процесор може цілком використовувати під час складних обчислень. Її розмір у демонстрованій імітаційній моделі складає 4 комірки по 24 біти. Існує можливість здійснювати запис та читання даних будь-якої комірки серед зазначених. Вміст оперативної пам’яті виводиться в термінал у шістнадцятковому вигляді (по 2 цифри на байт, по 3 байти на комірку).

## Команди

Усього було реалізовано 4 команди для роботи із пам’яттю (2 додаткові) та 2 різновиди команди індивідуального завдання:

Команди для роботи із пам’яттю:

* **Load R1 num** – надає **R1** регістру значення **num**
* **LoadOp R2 num** – вносить значення регістру **R2** до комірки з індексом **num**

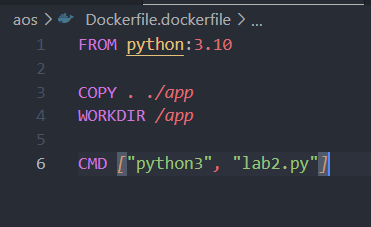
Різновиди команди індивідуального завдання:

* **LShift R3 num**  – виконує операцію логічного зсуву вліво на **num**  групи регістрів **R1-R3**
* **RShift R3 num**  – виконує операцію логічного зсуву вправо на **num**  групи регістрів **R1-R3**

# Використані методи

## Кросплатформенність

Також за використання інструмента *Docker* робота програми була протестована на віртуальній машині під керуванням операційної системи Ubuntu.

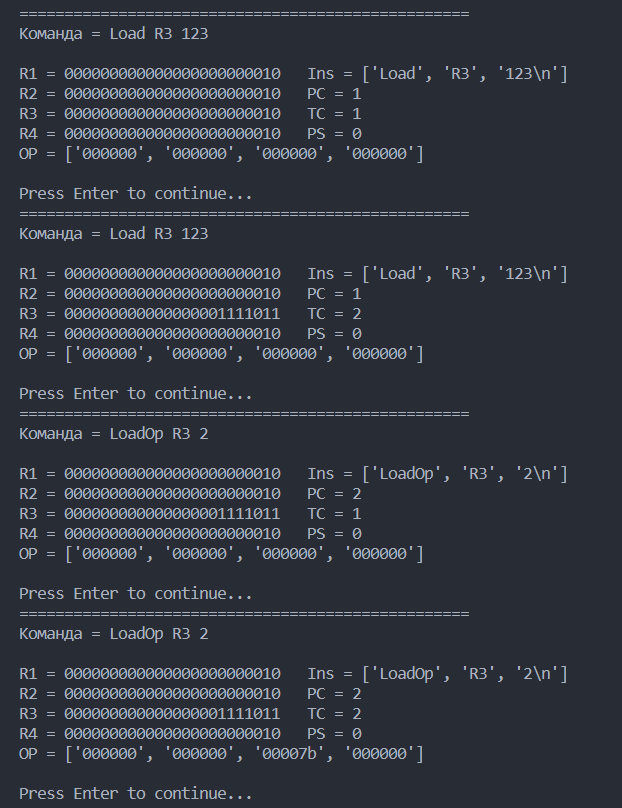


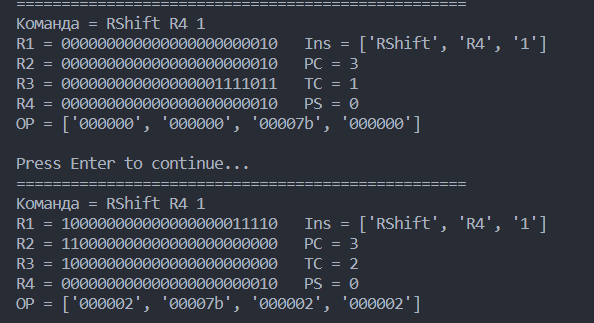
# Демонстрація роботи

## Програма

Результат роботи програми під час запуску в докері:

## Test2.txt

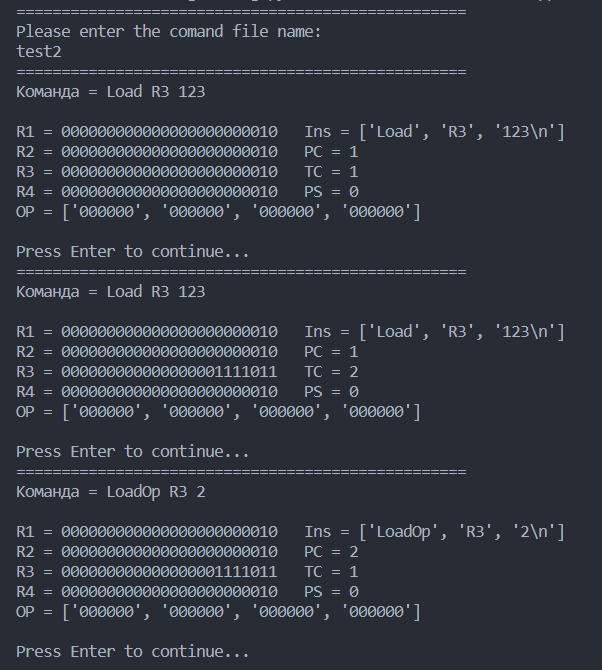


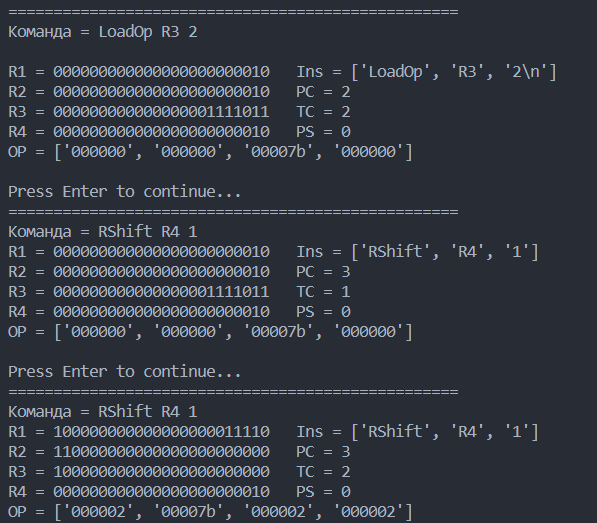


## Результат роботи в системі

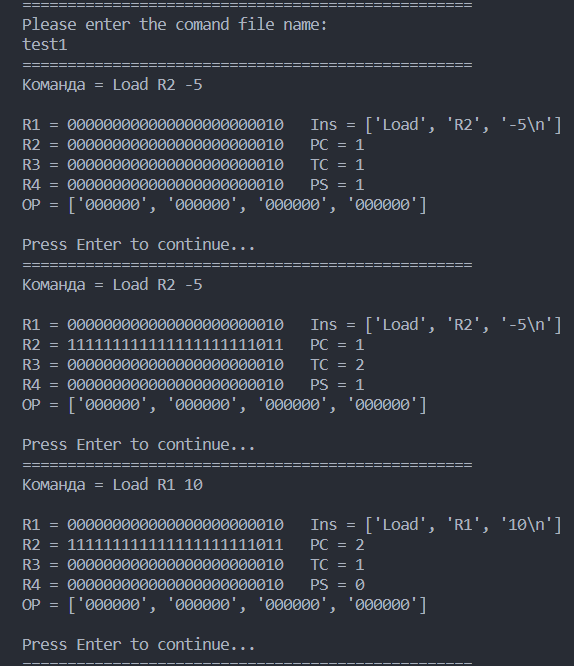
Результат роботи програми під час запуску в головній системі комп’ютера:

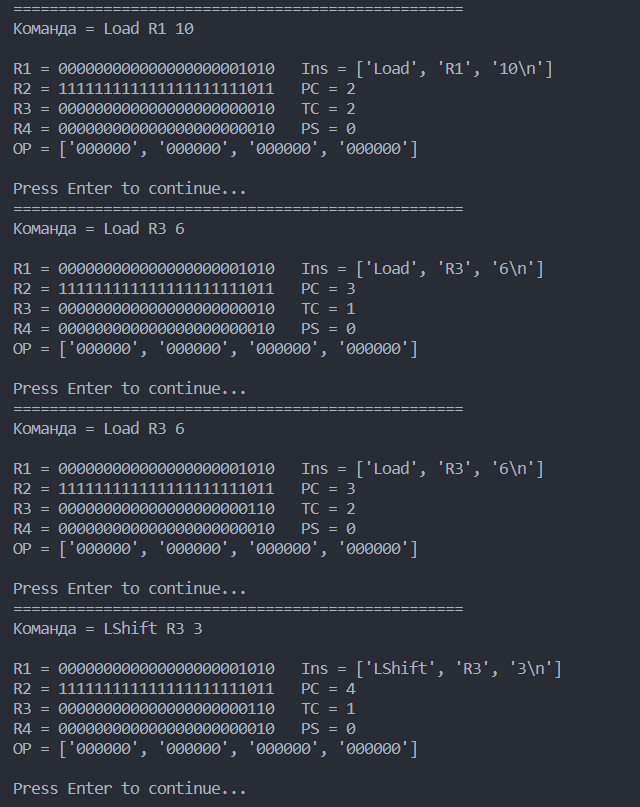
## Test2.txt

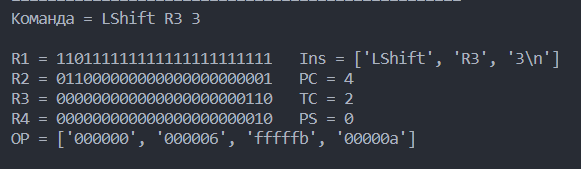




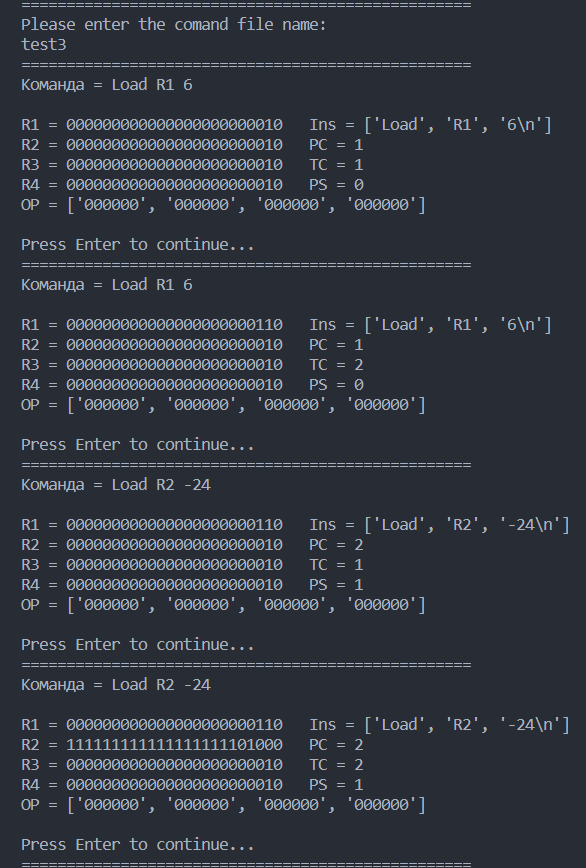
## Test1.txt

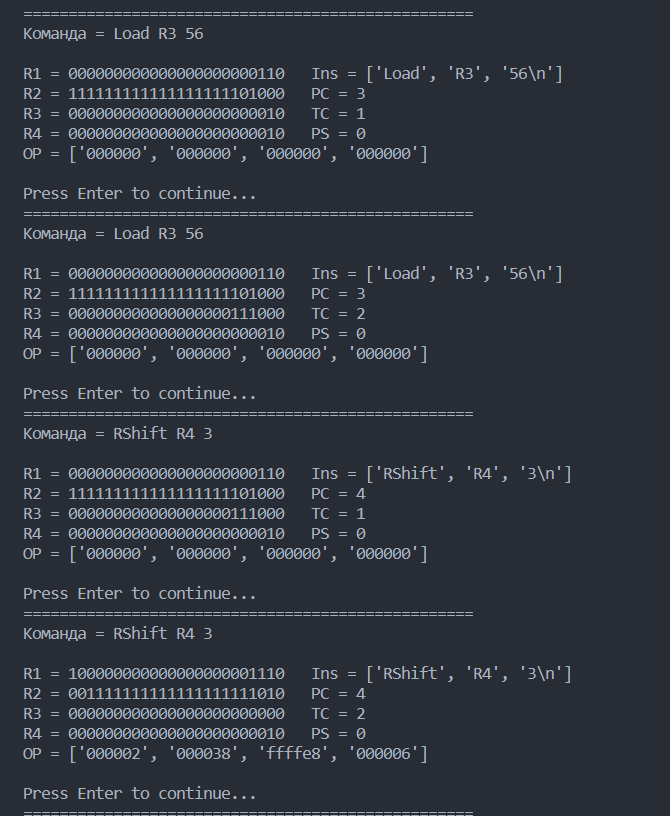


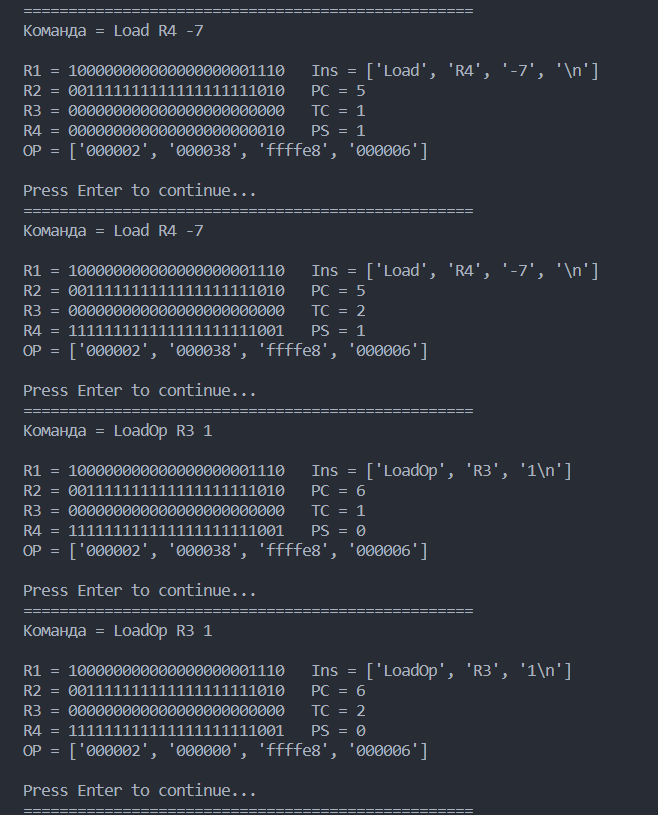


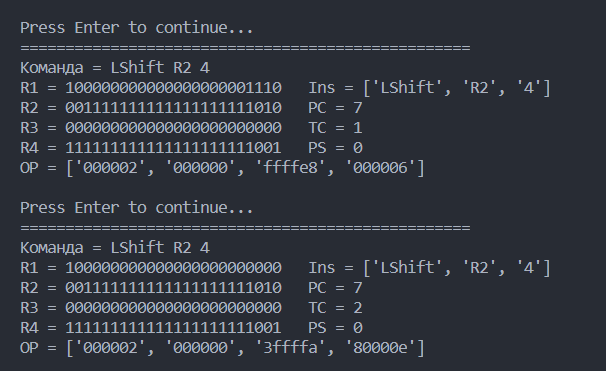


## Test3.txt









# Код програми

## Dockerfile

FROM python:3.10

COPY *.* *./app*

WORKDIR */app*

CMD *[*"python3"*,* "lab2.py"*]*

## Test1.txt

Load R2 -5

Load R1 10

Load R3 6

LShift R3 3

## Test2.txt

Load R3 123

LoadOp R3 2

RShift R4 1

## Test3.txt

Load R1 6

Load R2 -24

Load R3 56

RShift R4 3

Load R4 -7

LoadOp R3 1

LShift R2 4

## Lab2.py

*import* os

*import* platform

def fullBit(*binStr*):

*if* len(*binStr*) < 24:

*return* "0" \* (24-len(*binStr*)) + *binStr*

*return* *binStr*

def fullHex(*hexStr*):

*if* len(*hexStr*) < 6:

*return* "0" \* (6-len(*hexStr*)) + *hexStr*

*return* *hexStr*

def invertBin(*binStr*):

*binStr* = fullBit(*binStr*)

*return* ''.join('1' *if* x == '0' *else* '0' *for* x *in* *binStr*)

def suppCode(*binStr*):

    inv = invertBin(*binStr*)[::-1]

    overflow = 0

    res = []

    one = "1" + "0" \* (len(inv) - 1)

*for* obj *in* zip(inv, one):

        value = int(obj[0]) + int(obj[1]) + overflow

        overflow = value // 2

        res.append(value % 2)

*if* overflow == 1:

        res.append(1)

    res = res[::-1]

*return* ''.join(map(str, res))

def intToBin(*num*):

*try*:

*if* (int(*num*) <= 0):

            binStr = bin(int(*num*))[3:]

*return* suppCode(binStr)

        binStr = bin(int(*num*))[2:]

*return* fullBit(binStr)

*except*:

*raise* Exception("Invalid number")

def binToHex(*num*):

*return* hex(int(*num*, 2))[2:]

def hexToBin(*num*):

*return* fullBit(bin(int(*num*, 16))[2:])

def load(*reg*, *num*):

    global R1, R2, R3, R4

*match* *reg*:

*case* "R1":

            R1 = intToBin(*num*)

*case* "R2":

            R2 = intToBin(*num*)

*case* "R3":

            R3 = intToBin(*num*)

*case* "R4":

            R4 = intToBin(*num*)

*case* \_:

*raise* Exception("Invalid register")

def loadOp(*reg*, *num*):

    global R1, R2, R3, R4, OP

*try*:

*num* = int(*num*)

*if* (*num* > 3 *or* *num* < 0):

*raise* Exception("Invalid index of OP")

*except*:

*raise* Exception("Invalid index of OP")

*match* *reg*:

*case* "R1":

            OP[*num*] = fullHex(binToHex(R1))

*case* "R2":

            OP[*num*] = fullHex(binToHex(R2))

*case* "R3":

            OP[*num*] = fullHex(binToHex(R3))

*case* "R4":

            OP[*num*] = fullHex(binToHex(R4))

*case* \_:

*raise* Exception("Invalid register")

def shift(*reg*, *num*, *direct*):

    global R1, R2, R3, R4, OP

    arr = ["R1", "R2", "R3", "R4"]

    j = 3

    l = 0

*for* i *in* arr:

*match* i:

*case* "R1":

                OP[j] = fullHex(binToHex(R1))

*case* "R2":

                OP[j] = fullHex(binToHex(R2))

*case* "R3":

                OP[j] = fullHex(binToHex(R3))

*case* "R4":

                OP[j] = fullHex(binToHex(R4))

*if* i == *reg*:

*break*

        l += 1

        j -= 1

    bitStr = ""

*for* k *in* OP:

        bitStr += hexToBin(k)

*if* *direct* == "l":

        newStr = bin(int(bitStr, 2) << int(*num*))[2:]

*elif* *direct* == "r":

        newStr = bin(int(bitStr, 2) >> int(*num*))[2:]

*for* i *in* range(l):

*match* arr[i]:

*case* "R1":

                R1 = fullBit(newStr[24 \* 0: 24 \* (0 + 1)])

*case* "R2":

                R2 = fullBit(newStr[24 \* 1: 24 \* (1 + 1)])

*case* "R3":

                R3 = fullBit(newStr[24 \* 2: 24 \* (2 + 1)])

*case* "R4":

                R4 = fullBit(newStr[24 \* 3: 24 \* (3 + 1)])

def controller(*ArrComm*):

*match* *ArrComm*[0]:

*case* "Load":

            load(*ArrComm*[1], *ArrComm*[2])

*case* "LoadOp":

            loadOp(*ArrComm*[1], *ArrComm*[2])

*case* "LShift":

            shift(*ArrComm*[1], *ArrComm*[2], "l")

*case* "RShift":

            shift(*ArrComm*[1], *ArrComm*[2], "r")

*case* \_:

*raise* Exception("Invalid command")

def draw(*line*, *arr*):

    print("==================================================")

    print(f"Команда = {*line*}")

    print(f"R1 = {R1}   Ins = {*arr*}")

    print(f"R2 = {R2}   PC = {PC}")

    print(f"R3 = {R3}   TC = {TC % 2 + 1}")

    print(f"R4 = {R4}   PS = {PS}")

    print(f"OP = {OP}\n")

def openFile(*file*):

    global PC, TC, PS

    f = open(f"{*file*}", "r")

*for* line *in* f:

        arr = line.split(" ")

        TC += 1

        PC += 1

*if* int(arr[2]) > 0:

            PS = 0  *# "+"*

*else*:

            PS = 1  *# "-"*

        draw(line, arr)

        input("Press Enter to continue...")

        controller(arr)

        TC += 1

        draw(line, arr)

        input("Press Enter to continue...")

def main():

    global R1, R2, R3, R4, PC, TC, PS, OP

    binRand = intToBin(2)  *# random start number for register*

    R1, R2, R3, R4 = binRand, binRand, binRand, binRand  *# registers*

    PC = 0  *# command number*

    TC = 1  *# tact*

    PS = 0  *# sign*

    OP = ["0"\*6, "0"\*6, "0"\*6, "0"\*6]  *# 12 bytes RAM*

    print("================================================== ")

    print("Please enter the comand file name:")

*if* platform.system() == "Linux":

        fileName = os.path.dirname(\_\_file\_\_) + '/' + input() + ".txt"

*else*:

        fileName = os.path.dirname(\_\_file\_\_) + '\\' + input() + ".txt"

    openFile(fileName)

main()