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, 3, 7

Студента другого курсу
групи К-23(2)
Міщука Романа Андрійовича
Факультету комп’ютерних наук
та кібернетики

Київ – 2022

1. Постановка задачі

1.1. 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).

1.2. Задачі варіанту

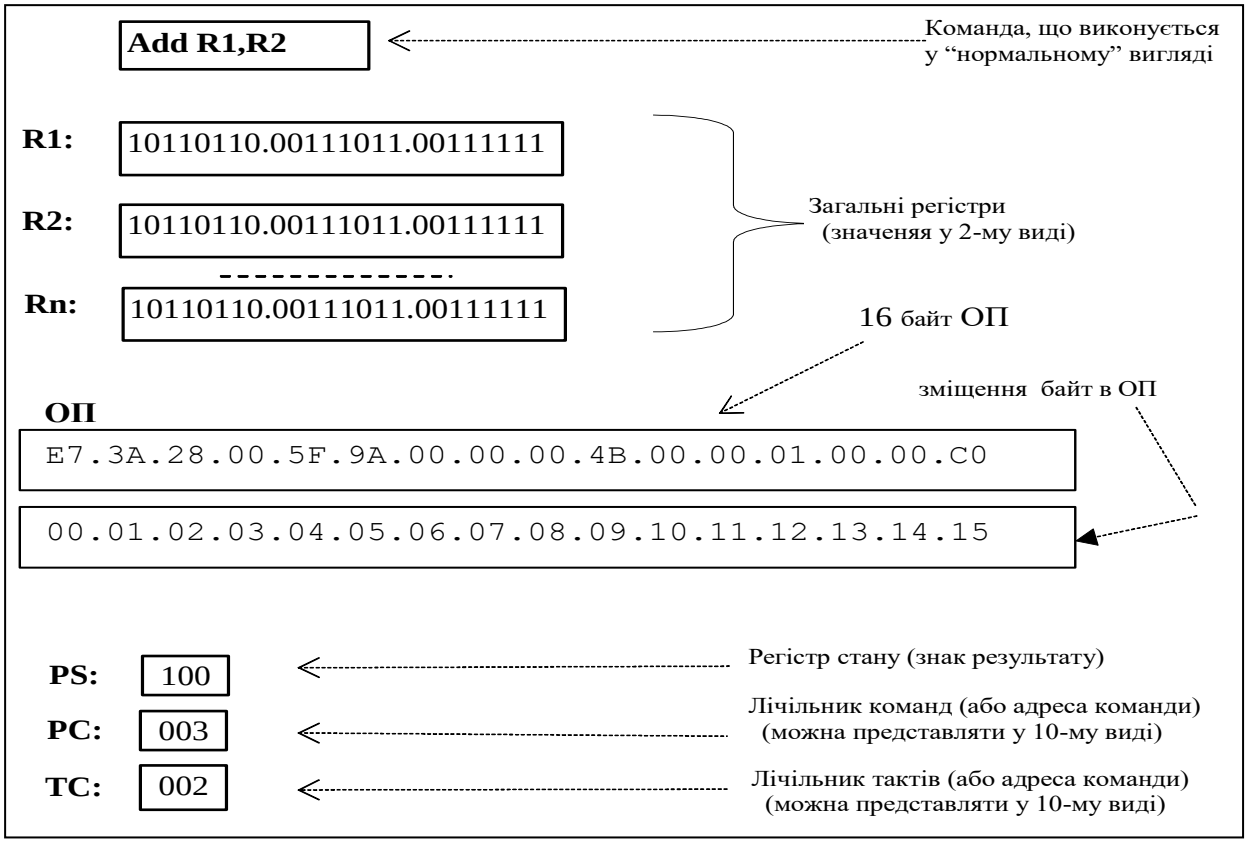
Індивідуальний варіант завдання: 1, 3, 7.

1) 1-й операнд завжди в акумуляторі, результат команди заноситься в акумулятор;
2) Бітність регістрів/стеку та операндів команд: 15 біт;
3) Індивідуальна команда процесора: X mod Y (остача від ділення X на Y).

1.3. 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:



Якщо має бути одно адресна безстекова модель команд, то ще потрібний регістр, який називають акумулятор. В командах він явно операндом не задається, хоча може використовуватись як реальний операнд. Типово для команд із акумулятором-операндом є зберігати результат команди саме в акумуляторі.

2. Реалізація

2.1. Регістри та пам'ять

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

- **R1** – регістр-акумулятор (відображається у двійковому вигляді);
- **IR** – регістр команди, що наразі виконується (відображається у вигляді рядка команди);
- **PC** – лічильник команд (порядковий номер рядка команди) (відображається у десятковому вигляді);
- **TC** – лічильник тактів (відображається у десятковому вигляді);
- **RS** – 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 10 комірок по 15 біт (з округленням вверх на відповідну кількість байт), із можливістю швидкого збільшення цієї кількості до будь-якого іншого значення. 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), разом із рядком байтового та коміркового зсуву.

2.2. 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:

- **`set_c`** `const` – надає R1 значення const;
- **`load_ca addr`** – надає R1 значення комірки оперативної пам’яті з номером addr.
- **`unwrap _`** – розіменування R1. Зберігає в R1 значення комірки з номером, який збігається зі значенням R1 (R1 = RAM[R1]). Ця функція не потребує операнда, і його значення буде ігноруватися;
- **`dump_ca addr`** – надає комірці оперативної пам’яті з номером addr значення R1;

Різновиди команди індивідуального завдання:

- **`mod_c const`** – виконує операцію R1 = R1 % const;
- **`mod_ca addr`** – виконує операцію R1 = R1 % RAM[addr];

- **`#`** або пустий рядок – рядки програми, що розпочинаються таким чином, ігноруються інтерпретатором команд, і відкидаються із загального списку.

3. Використані методи

3.1. Реалізація пам’яті

Для того щоб імітувати справжню пам’ять комп’ютера, використовується масив елементів із найменшим розміром, які при цьому можуть в себе включати одну комірку (отже використовується змінна розміром 2 байти). Через те, що кожна комірка становить нецілу кількість байт, то для реалізації пам’яті довелось використовувати побітові операції:

```
class bitmem
{
public:
    static constexpr size_t item_size = 15;

    typedef unsigned short mem_t;
    static constexpr size_t mem_t_size = sizeof(mem_t) * 8;

    static constexpr mem_t item_mask = (1 << item_size) - 1;
    static constexpr int char_mask = (1 << 8) - 1;

private:
    mem_t* mem;
    size_t mem_size;
...

// Setting the current and next cell:
//      |......item|..shift|
// |mem_T 1 ...|mem_T 0 ...|
bitmem::mem_t bitmem::get(const size_t i) const
{
       const size_t bit_index = i * item_size;
       const size_t index = bit_index / mem_t_size;
       const size_t bit_shift = bit_index % mem_t_size;

       mem_t ret = mem[index] >> bit_shift;
       if (bit_shift) ret |= mem[index + 1] << (mem_t_size - bit_shift);
       ret &= item_mask;
       return ret;
}
void bitmem::set(const size_t i, const mem_t val)
{
       const size_t bit_index = i * item_size;
       const size_t index = bit_index / mem_t_size;
       const size_t shift = bit_index % mem_t_size;
       mem[index] = (mem_t)(val << shift) |
              (mem[index] & ((mem_t)(1 << shift) - 1));

       if (shift > 0) {
              mem[index + 1] = (mem[index + 1] & (
                     (1 << mem_t_size) - (1 << (item_size + shift - mem_t_size)))
                ) |
                ((mem_t)val >> (mem_t_size - shift));
       }
}
```

3.2. Кросплатформенність

За допомогою інструменту *CMake* була реалізована можливість легкої компіляції коду під різні платформи та компілятори.

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

4. Демонстрація роботи

4.1. Програма

Для демонстрації роботи була написана невеличка програма, яка рахує значення виразу із %. Вона має ось такий вигляд:

```
# Computing (a % c) % (c % b)
#
# a = 32048
# b = 287
# c = 7298

# store a, b, c
set_c 32048
dump_ca 0
set_c 287
dump_ca 1
set_c 7298
dump_ca 2

# c % b
load_ca 2
mod_ca 1
dump_ca 3

# a % c
load_ca 0
mod_ca 2

# result
mod_ca 3
```

При цьому в зазначеному коді використовуються більшість із реалізованих команд.

4.2. Результат роботи в системі

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

```
Total program length: 12 lines

Starting debug:
-----------------------------------
IR = set_c 1111101.00110000

R1 = 0000000.00000000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 0
```

```
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = set_c 1111101.00110000

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 0
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = dump_ca 0000000.00000000

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 1
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = dump_ca 0000000.00000000

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 1
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = set_c 0000001.00011111

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 2
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = set_c 0000001.00011111

R1 = 0000001.00011111

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 2
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = dump_ca 0000000.00000001
```

```
R1 = 0000001.00011111

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 3
TC = 0
RS = 0000000.00000000
-------------------------------------
IR = dump_ca 0000000.00000001

R1 = 0000001.00011111

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 3
TC = 1
RS = 0000000.00000000
-------------------------------------
-------------------------------------
IR = set_c 0011100.10000010

R1 = 0000001.00011111

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 4
TC = 0
RS = 0000000.00000000
-------------------------------------
IR = set_c 0011100.10000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 4
TC = 1
RS = 0000000.00000000
-------------------------------------
-------------------------------------
IR = dump_ca 0000000.00000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 5
TC = 0
RS = 0000000.00000000
-------------------------------------
IR = dump_ca 0000000.00000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00
```

```
PC = 5
TC = 1
RS = 0000000.00000000
-------------------------------------
-------------------------------------
IR = load_ca 0000000.00000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 6
TC = 0
RS = 0000000.00000000
-------------------------------------
IR = load_ca 0000000.00000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 6
TC = 1
RS = 0000000.00000000
-------------------------------------
-------------------------------------
IR = mod_ca 0000000.00000001

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 7
TC = 0
RS = 0000000.00000000
-------------------------------------
IR = mod_ca 0000000.00000001

R1 = 0000000.01111011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 7
TC = 1
RS = 0000000.00000000
-------------------------------------
-------------------------------------
IR = dump_ca 0000000.00000011

R1 = 0000000.01111011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 8
TC = 0
RS = 0000000.00000000
-------------------------------------
IR = dump_ca 0000000.00000011
```

```
R1 = 0000000.01111011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 8
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = load_ca 0000000.00000000

R1 = 0000000.01111011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 9
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = load_ca 0000000.00000000

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 9
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = mod_ca 0000000.00000010

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 10
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = mod_ca 0000000.00000010

R1 = 0001011.00101000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 10
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = mod_ca 0000000.00000011

R1 = 0001011.00101000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
```

```
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 11
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = mod_ca 0000000.00000011

R1 = 0000000.00011011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 11
TC = 1
RS = 0000000.00000000
-----------------------------------
```

4.3. Результат роботи у віртуальному середовищі

Результат роботи програми у віртуальному середовищі Docker під керуванням операційної системи Ubuntu (компілятор GCC):

```
Total program length: 12 lines

Starting debug:
-----------------------------------
IR = set_c 1111101.00110000

R1 = 0000000.00000000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 0
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = set_c 1111101.00110000

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 0
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = dump_ca 0000000.00000000

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 1
TC = 0
RS = 0000000.00000000
-----------------------------------
```

```
IR = dump_ca 0000000.00000000

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 1
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = set_c 0000001.00011111

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 2
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = set_c 0000001.00011111

R1 = 0000001.00011111

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 2
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = dump_ca 0000000.00000001

R1 = 0000001.00011111

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 3
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = dump_ca 0000000.00000001

R1 = 0000001.00011111

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 3
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = set_c 0011100.10000010

R1 = 0000001.00011111

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.8f.fd.30
```

```
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 4
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = set_c 0011100.10000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 4
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = dump_ca 0000000.00000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 5
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = dump_ca 0000000.00000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 5
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = load_ca 0000000.00000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 6
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = load_ca 0000000.00000010

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 6
TC = 1
RS = 0000000.00000000
```

```
-----------------------------------
-----------------------------------
IR = mod_ca 0000000.00000001

R1 = 0011100.10000010

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 7
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = mod_ca 0000000.00000001

R1 = 0000000.01111011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 7
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = dump_ca 0000000.00000011

R1 = 0000000.01111011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 8
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = dump_ca 0000000.00000011

R1 = 0000000.01111011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 8
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = load_ca 0000000.00000000

R1 = 0000000.01111011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 9
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = load_ca 0000000.00000000

R1 = 1111101.00110000
```

```
RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 9
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = mod_ca 0000000.00000010

R1 = 1111101.00110000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 10
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = mod_ca 0000000.00000010

R1 = 0001011.00101000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 10
TC = 1
RS = 0000000.00000000
-----------------------------------
-----------------------------------
IR = mod_ca 0000000.00000011

R1 = 0001011.00101000

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 11
TC = 0
RS = 0000000.00000000
-----------------------------------
IR = mod_ca 0000000.00000011

R1 = 0000000.00011011

RAM          00.00.00.00.00.00.00.00.00.00.00.00.0f.67.20.80.8f.fd.30
Bytes offset 19.18.17.16.15.14.13.12.11.10.09.08.07.06.05.04.03.02.01.00
Items offset 10.09.09.08.08.07.06.06.05.05.04.04.03.03.02.02.01.01.00.00

PC = 11
TC = 1
RS = 0000000.00000000
-----------------------------------
```

5. Код програми

5.1. Dockerfile

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y g++ cmake make
RUN apt-get install -y libncurses5-dev libncursesw5-dev

COPY . /usr/src/LAB2
WORKDIR /usr/src/LAB2

RUN cmake -S . -B ./build
RUN cmake --build ./build

CMD ./build/main
```

5.2. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15)
project(LAB2_AES VERSION 1.0 LANGUAGES CXX)
set(CMAKE_CXX_STANDARD 14)

file("GLOB_RECURSE" "SOURCE_FILES" "src/*.cpp")
add_executable(main LAB2.cpp "include/bitmem.h" ${SOURCE_FILES})
target_include_directories(main PUBLIC "./include")

if(UNIX)
	target_link_libraries(main ncurses)
endif(UNIX)
```

5.3. program.txt

```
# Computing (a % c) % (c % b)
#
# a = 32048
# b = 287
# c = 7298

# store a, b, c
set_c 32048
dump_ca 0
set_c 287
dump_ca 1
set_c 7298
dump_ca 2

# c % b
load_ca 2
mod_ca 1
dump_ca 3

# a % c
load_ca 0
mod_ca 2

# result
mod_ca 3
```

5.4. LAB2.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <fstream>

#include "processor.h"

#ifdef _WIN32
#include <conio.h>
#endif
#ifdef linux
#include <curses.h>
#endif

using namespace std;

vector<string> read_file(string filename = "program.txt")
{
    ifstream file(filename, ios_base::in);
    vector<string> ret{};
    string buff;
    while(getline(file, buff))
    {
           if(buff.empty() || buff[0] == '#' || buff[0] == '\r') continue;
        ret.push_back(buff);
        //cout << buff << "\n";
    }
    file.close();
    return ret;
}

int main()
{
    auto program = read_file();
    processor p(program);
    cout << p.get_program_info() << "\nStarting debug: \n";

    while(p.do_tick())
    {
        cout << p.get_state();
#ifdef _WIN32
        _getch();
#endif
#ifdef linux
        getch();
#endif
        p.end_tick();
    }
}
```

5.5. processor.h

```
#pragma once
#include <vector>
#include <string>
#include <sstream>
#include <map>

#include "bitmem.h"

class processor
{
public:
    using reg_t = bitmem::mem_t;

private:
    static const reg_t tc_num = 2;

    reg_t R1;
    reg_t PC, TC, RS;
    bitmem ram;

    struct command
    {
        enum name_t : reg_t
        {
            set_c, load_ca, unwrap, dump_ca, mod_c, mod_ca,
            COUNT
        };
        static std::map<name_t, const char*> name_map;

        name_t name;
        reg_t val;

        std::string str() const;
    } IR;

    std::vector<std::string> program;

public:
    processor(std::vector<std::string> program, size_t ram_size = 10);

    bool do_tick();
    void end_tick();

    std::string get_state() const;

    std::string get_program_info() const;
};
```

5.6. processor.cpp

```
#include "processor.h"

#include <cstring>
#include <iomanip>
#include <algorithm>

using namespace std;

std::map<processor::command::name_t, const char*> processor::command::name_map{
	{set_c, "set_c"},
	{load_ca, "load_ca"},
	{unwrap, "unwrap"},
	{dump_ca, "dump_ca"},
	{mod_c, "mod_c"},
	{mod_ca, "mod_ca"},
};

string to_binary(processor::reg_t n)
{
	stringstream s;
	while (n > 0)
	{
		s << n % 2;
		n >>= 1;
	}
	string ret = s.str();
	reverse(ret.begin(), ret.end());
	ret = string(bitmem::item_size - ret.size(), '0') + ret;
	for (int i = bitmem::item_size - 8; i > 0; i -= 8)
		ret.insert(i, 1, '.');
	return ret;
}

std::string processor::command::str() const
{
	stringstream ret;
	ret << name_map[name] << " " << to_binary(val);
	return ret.str();
}

processor::processor(vector<string> program, const size_t ram_size):
	ram(ram_size), R1(0), PC(0), TC(0), RS(0), IR()
{
	this->program = move(program);
}

bool processor::do_tick()
{
	if (PC >= program.size()) return false;

	if (TC == 0)
	{
		// Parsing command data from string
		stringstream s;
		s << program[PC];

		string str_name;
		s >> str_name;
		reg_t name = 0;
		while (
			strcmp(str_name.c_str(), command::name_map[(command::name_t)name]) != 0
&&
			name < command::name_t::COUNT
		) name++;
```

```
                IR.name = (command::name_t) name;
                if(IR.name != command::unwrap)
                        s >> IR.val;
        }
        else if (TC == 1)
        {
                switch (IR.name)
                {
                case command::set_c:
                        R1 = IR.val;
                        break;
                case command::dump_ca:
                        ram.set(IR.val, R1);
                        break;
                case command::load_ca:
                        R1 = ram.get(IR.val);
                        break;
                case command::unwrap:
                        R1 = ram.get(R1);
                        break;
                case command::mod_c:
                        R1 %= IR.val;
                        break;
                case command::mod_ca:
                        R1 %= ram.get(IR.val);
                        break;
                }
        }

        return PC < program.size();
}
void processor::end_tick()
{
        if (TC == tc_num - 1) PC++;
        TC++;
        TC %= tc_num;
}

std::string processor::get_state() const
{
        static constexpr auto delim = "-----------------------------------\n";

        stringstream ss;
        ss << delim << "IR = " << IR.str() << "\n\n";

        ss << "R1 = " << to_binary(R1) << "\n\n";

        ss << "RAM          " << ram.str() << "\n";
        ss << "Bytes offset " << ram.byte_offset() << "\n";
        ss << "Items offset " << ram.item_offset() << "\n\n";

        ss << "PC = " << PC << "\n";
        ss << "TC = " << TC << "\n";
        ss << "RS = " << to_binary(RS) << "\n";

        if (TC == tc_num - 1) ss << delim;

        return ss.str();
}

std::string processor::get_program_info() const
{
        stringstream ret;
        ret << "Total program length: " << program.size() << " lines\n";
        return ret.str();
}
```

5.7. bitmem.h

```h
#pragma once

#include <string>
#include <sstream>

class bitmem
{
public:
    static constexpr size_t item_size = 15;

    typedef unsigned short mem_t;
    static constexpr size_t mem_t_size = sizeof(mem_t) * 8;

    static constexpr mem_t item_mask = (1 << item_size) - 1;
    static constexpr int char_mask = (1 << 8) - 1;

private:
    mem_t* mem;
    size_t mem_size;

public:

    bitmem(size_t items_num);
    ~bitmem() noexcept;

    mem_t get(size_t i) const;
    void set(size_t i, mem_t val);

    std::string str() const;
    std::string byte_offset() const;
    std::string item_offset() const;
};
```

5.8. bitmem.cpp

```
#include "bitmem.h"

#include <algorithm>
#include <iomanip>

using namespace std;

bitmem::bitmem(size_t items_num)
{
	mem_size = items_num * item_size;
	mem_size = mem_size / mem_t_size + (mem_size % mem_t_size != 0);
	mem = new mem_t[mem_size];
	std::fill(mem, mem + mem_size, 0);
}
bitmem::~bitmem() noexcept
{
	delete[] mem;
}

// Setting the current and next cell:
//      |......item|..shift|
// |mem_T 1 ...|mem_T 0 ...|
bitmem::mem_t bitmem::get(const size_t i) const
{
	const size_t bit_index = i * item_size;
	const size_t index = bit_index / mem_t_size;
	const size_t bit_shift = bit_index % mem_t_size;

	mem_t ret = mem[index] >> bit_shift;
	if (bit_shift) ret |= mem[index + 1] << (mem_t_size - bit_shift);
	ret &= item_mask;
	return ret;
}
void bitmem::set(const size_t i, const mem_t val)
{
	const size_t bit_index = i * item_size;
	const size_t index = bit_index / mem_t_size;
	const size_t shift = bit_index % mem_t_size;
	mem[index] = (mem_t)(val << shift) |
		(mem[index] & ((mem_t)(1 << shift) - 1));

	if (shift > 0) {
		mem[index + 1] = (mem[index + 1] & (
				(1 << mem_t_size) - (1 << (item_size + shift -
mem_t_size)))
			) |
			((mem_t)val >> (mem_t_size - shift));
	}
}

std::string bitmem::str() const
{
	stringstream s;
	auto* buff = (unsigned const char*)mem;
	for (size_t i = mem_size * mem_t_size / 8 - 1; i >= 1; i--)
	{
		s << hex << setfill('0') << setw(2) << (int)buff[i] << '.';
	}
	s << hex << setfill('0') << setw(2) << (int)buff[0];
	return s.str();
}
std::string bitmem::byte_offset() const
{
	stringstream s;
	for (size_t i = mem_size * mem_t_size / 8 - 1; i >= 1; i--)
```

```
	{
		s << setfill('0') << setw(2) << i << '.';
	}
	s << hex << setfill('0') << setw(2) << 0;
	return s.str();
}
std::string bitmem::item_offset() const
{
	stringstream s;
	for (size_t i = mem_size * mem_t_size / 8 - 1; i >= 1; i--)
	{
		s << setfill('0') << setw(2) << i * 8 / item_size << '.';
	}
	s << hex << setfill('0') << setw(2) << 0;
	return s.str();
}
```