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

# 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: 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).

## 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.

# Реалізація

## 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.

## 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.

# Використані методи

## 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));

}

}

## Кросплатформенність

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

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

# Демонстрація роботи

## 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.

## 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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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

------------------------------------

## 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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.7d.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.00.07.20.80.8f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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 00.00.00.00.00.00.00.00.00.00.00.00.00.0f.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

------------------------------------

# Код програми

## 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

## 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)

## 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

## LAB2.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();

}

}

## 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;

};

## 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();

}

## bitmem.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;

};

## 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();

}