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

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

2.77121e+08

+ int 2.23098e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 81%

- int 2.56086e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 92%

\* int 2.56229e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 92%

/ int 2.59154e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 94%

+ long 2.69418e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 97%

- long 1.59385e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXX 58%

\* long 2.43425e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 88%

/ long 2.44797e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 88%

+ llong 2.48866e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 90%

- llong 2.77121e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 100%

\* llong 2.70267e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 98%

/ llong 2.46716e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 89%

+ char 2.05071e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 74%

- char 2.65877e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 96%

\* char 2.69362e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 97%

/ char 2.62953e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 95%

+ float 2.67815e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 97%

- float 2.60259e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 94%

\* float 2.59863e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 94%

/ float 2.49151e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 90%

+ double 2.60073e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 94%

- double 2.65449e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 96%

\* double 2.62868e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 95%

/ double 2.38984e+08 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 86%

# Код програми

## Dockerfile

FROM ubuntu:latest

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

COPY . /usr/src/LAB1

WORKDIR /usr/src/LAB1

RUN cmake -S . -B ./build

RUN cmake --build ./build

CMD ./build/main

## CMakeLists.txt

cmake\_minimum\_required(VERSION 3.15)

project(LAB1\_AES VERSION 1.0 LANGUAGES CXX)

set(CMAKE\_CXX\_FLAGS\_RELEASE "-O2")

add\_executable(main LAB1.cpp)

## LAB1.cpp

#include <iostream>

#include <chrono>

#include <tuple>

#include <vector>

#include <string>

#include <iomanip>

#include <cmath>

using namespace std;

#if defined(\_\_GNUC\_\_) || defined(\_\_GNUG\_\_)

#define in\_void inline void \_\_attribute\_\_((always\_inline))

#else

#define in\_void \_\_forceinline void

#endif

enum operation

{

addition,

subtraction,

multiplication,

division,

no\_operation

};

template <operation T>

using operation\_t = integral\_constant<operation, T>;

constexpr operation\_t<no\_operation> no\_operation\_constant{};

template <typename T>

using op\_arg\_t = volatile T;

template<typename T>

in\_void exec\_operation(op\_arg\_t<T> a, op\_arg\_t<T> b, volatile T& res, operation\_t<addition>)

{ res = a + b; }

template<typename T>

in\_void exec\_operation(op\_arg\_t<T> a, op\_arg\_t<T> b, volatile T& res, operation\_t<subtraction>)

{ res = a - b; }

template<typename T>

in\_void exec\_operation(op\_arg\_t<T> a, op\_arg\_t<T> b, volatile T& res, operation\_t<multiplication>)

{ res = a \* b; }

template<typename T>

in\_void exec\_operation(op\_arg\_t<T> a, op\_arg\_t<T> b, volatile T& res, operation\_t<division>)

{ res = a / b; }

template<typename T>

in\_void exec\_operation(op\_arg\_t<T> a, op\_arg\_t<T> b, volatile T& res, operation\_t<no\_operation>)

{

res = a;

//res = b;

}

typedef unsigned long long \_loop\_t;

enum base\_using { yes, no };

template <base\_using T>

using base\_using\_t = integral\_constant<base\_using, T>;

template <typename T>

using rep\_arg\_t = volatile T&;

template<typename T, operation O, const \_loop\_t R, enable\_if\_t<0 == R, bool> = 0>

in\_void exec\_repeated(rep\_arg\_t<T> a, rep\_arg\_t<T> b, volatile T& res, const operation\_t<O>& op, const base\_using\_t<no>)

{

exec\_operation(a, b, res, op);

}

template<typename T, operation O, const \_loop\_t R, enable\_if\_t<0 == R, bool> = 0>

in\_void exec\_repeated(rep\_arg\_t<T> a, rep\_arg\_t<T> b, volatile T& res, const operation\_t<O>& op, const base\_using\_t<yes>)

{

exec\_operation(a, b, res, op);

exec\_operation(a, b, res, no\_operation\_constant);

}

template<typename T, operation O, const \_loop\_t R, base\_using B, enable\_if\_t<0 < R, bool > = 0>

in\_void exec\_repeated(rep\_arg\_t<T> a, rep\_arg\_t<T> b, volatile T& res, const operation\_t<O>& op, const base\_using\_t<B>& base)

{

exec\_repeated<T, O, 0>(a, b, res, op, base);

exec\_repeated<T, O, R-1, B>(a, b, res, op, base);

}

template<typename T, operation O, const \_loop\_t R, base\_using B>

double measure\_time(rep\_arg\_t<T> a, rep\_arg\_t<T> b, volatile T& res, const operation\_t<O>& op, const base\_using\_t<B>& base)

{

chrono::high\_resolution\_clock::time\_point begin{}, end{};

begin = chrono::high\_resolution\_clock::now();

exec\_repeated<T, O, R>(a, b, res, op, base);

end = chrono::high\_resolution\_clock::now();

return chrono::duration\_cast<chrono::nanoseconds>(end - begin).count();

}

#pragma optimize( "", off )

#pragma GCC push\_options

#pragma GCC optimize ("O0")

template <typename T, operation O, const \_loop\_t R>

double run\_test(const \_loop\_t count = 100)

{

constexpr base\_using\_t<yes> op\_based{};

constexpr base\_using\_t<no> op\_not\_based{};

const operation\_t<O> op{};

double ret = 0, buff;

T res = 0, a = 1, b = 1;

for (\_loop\_t i = 0; i < count; i++) {

buff = 2 \* measure\_time<T, O, R>(a, b, res, op, op\_not\_based) -

measure\_time<T, O, R>(a, b, res, op, op\_based);

if(buff < 0)

{

i--;

continue;

}

ret += buff;

}

return R \* count \* 1e9 / ret;

}

#pragma GCC pop\_options

#pragma optimize( "", on )

template<typename T, const \_loop\_t R, const \_loop\_t C>

tuple<string, double> run\_test\_for(const operation O)

{

switch (O)

{

case addition:

return {"+", run\_test<T, addition, R>(C)};

case subtraction:

return {"-", run\_test<T, subtraction, R>(C)};

case multiplication:

return {"\*", run\_test<T, multiplication, R>(C)};

case division:

return {"/", run\_test<T, division, R>(C)};

default:

return {"", 0};

}

}

template<const \_loop\_t R, const \_loop\_t C>

tuple<string, string, double> run\_test\_for(const int type\_id, const operation O)

{

switch (type\_id)

{

case 0:

return tuple\_cat(make\_tuple("int"), run\_test\_for<int, R, C>(O));

case 1:

return tuple\_cat(make\_tuple("long"), run\_test\_for<long, R, C>(O));

case 2:

return tuple\_cat(make\_tuple("llong"), run\_test\_for<long long, R, C>(O));

case 3:

return tuple\_cat(make\_tuple("char"), run\_test\_for<char, R, C>(O));

case 4:

return tuple\_cat(make\_tuple("float"), run\_test\_for<float, R, C>(O));

case 5:

return tuple\_cat(make\_tuple("double"), run\_test\_for<double, R, C>(O));

default:

return tuple\_cat(make\_tuple(""), run\_test\_for<int, R, C>(O));

}

}

void print\_result(const double max, const tuple<string, string, double>& result)

{

constexpr int bar\_len = 50;

cout << setw(2) << std::left << std::get<1>(result)

<< setw(10) << std::left << std::get<0>(result)

<< setw(16) << std::left << std::get<2>(result)

<< setw(bar\_len + 1) << std::left << string(round(std::get<2>(result) \* bar\_len / max), 'X')

<< round(std::get<2>(result) \* 100 / max) << "%\n";

}

constexpr int types\_num = 6, ops\_num = 4;

void take\_measurements(vector<vector<tuple<string, string, double>>> &results, double& max, double& avg)

{

constexpr \_loop\_t R = 100, C = 1e5;

constexpr double infty = numeric\_limits<double>::infinity();

max = avg = 0;

results = vector<vector<tuple<string, string, double>>>(types\_num, vector<tuple<string, string, double>>(ops\_num));

tuple<string, string, double> buff\_result{};

double& time = std::get<2>(buff\_result);

for (int type = 0; type < types\_num; type++)

for (int op = 0; op < ops\_num; op++) {

time = 0;

while (time <= 0 || !(time < infty)) {

buff\_result = run\_test\_for<R, C>(type, (operation)op);

}

results[type][op] = buff\_result;

if (time > max) max = time;

avg += time;

}

avg /= types\_num \* ops\_num;

}

int main()

{

cout.precision(6);

double max = 1e9, avg = 1;

vector<vector<tuple<string, string, double>>> results;

// Running tests till system is stable

while (max / avg > 5) {

take\_measurements(results, max, avg);

}

cout << max << "\n";

for (int type = 0; type < types\_num; type++) {

for (int op = 0; op < ops\_num; op++)

print\_result(max, results[type][op]);

cout << "\n";

}

return 0;

}