

# Звіт

до лабораторної роботи №1 на тему:  
*«Визначення швидкодії обчислювальної системи»*

*Студентки другого курсу*

*групи К-23/1*

*Факультету комп'ютерних наук*

*та кібернетики*

***Гудзенко Олександри***

Київ-2022

## Мета

Лабораторна робота передбачає розробку спрощеної системи тестів. Необхідно розробити програму, яка вимірює кількість виконуваних базових операцій (команд) за секунду конкретною Обчислювальною Системою (комп'ютер + ОС + Система програмування). Вимірювання "чистої" команди процесора не потрібне (як і є у реальних програмних комплексах, що типово розробляються на мовах високого рівня, часто навіть на платформенно незалежних).

## Основні принципи виконання роботи

До базового набору операцій достатньо включити операції додавання, віднімання, множення та ділення для кожного з базових типів даних (символьний, варіанти цілого, дійсний тощо, як це є в тій чи тій мові чи системі програмування). Інші операції, команди та типи – за бажанням. Наприклад, для C/C++ потрібно взяти типи `char`, `int`, `long`, `float` та `double`. Враховуючи, що для всіх типів процесорів характерне апаратне об'єднання команди додавання та віднімання у одну команду за рахунок відповідної зміни знаку одного з операндів (але для зручності кодування на рівні командного набору ці операції фігурують як окремі команди), не можна проводити вимірювання лише для однієї з цих двох команд. Якраз на цих двох операціях легко побачити стабільність тесту. Ви побачите, що за формальною, здавалось би, подібністю додавання та віднімання, добитися однакового результату непроста задача і потребує деяких "тонких" моментів при програмуванні.

Точність вимірювання - 2%. Достатньо на рівні вимірювань для лабораторної вважати, що для коротких операцій (додавання/віднімання для цілих слів) кількість операцій приблизно відповідає тактовій частоті процесора комп'ютера. Від цієї величини беремо 2%, і це значення буде  $\pm$ похибка між однойменними результатами для серії запусків програми. Наприклад, при тактовій частоті у 2 ГГц, 2% дорівнюватиме 40 МГц, або це приблизно 40 млн. коротких оп/сек; отже похибка між однойменними результатами  $\pm 40$  млн. оп/сек. Тобто самий швидкий результат (наприклад для операції додавання) беремо за 100%, а подібна операція (тоді нехай віднімання) може відстати від додавання на 40 млн. Операцій.

Програма має демонструвати стабільність вимірювань для серії запусків. Потрібно враховувати, що при роботі на платформі MS Windows під час вимірювання за тестом може початися процес свопінгу системи, тому в такому випадку сусідні вимірювання у серії можуть відрізнятися на порядок. Такі запуски потрібно виключати з розгляду.

Результати мають бути представлені у табличній формі з відображенням для кожного тесту:

1. назви команди/операції,
2. типу/формату даних,
3. кількості операцій за секунду (зайві знаки у мантисі для заданої точності не відображати),
4. лінійної діаграми значення швидкості у відсотках відносно самої швидкої команди/операції, яка береться за 100%,
5. значення у відсотках (можна округляти до цілого).

Для текстового режиму всі результати повинні поміститися на одній сторінці і для нормальної читабельності вирівняні у колонках.

Виділення кольором, різними шрифтами тощо – зайве; у лабораторній важлива якість тестування, що і є метою роботи. Всі інші ефекти – за бажанням виконавця, але враховується найперше якість тесту, і розвинений інтерфейс не може замінити точність тесту.

Тест не повинен довго виконуватися; нормально - до 1 хвилини для всіх вимірювань. При правильному проектуванні тесту та програми має бути локалізованим необхідний діапазон кількості операцій для заданої точності. Збільшення ж числа ітерацій для похибки у 2% буде зайвою, а для мультипрограмних середовищ навіть призведе до її накопичення.

## Виконання роботи

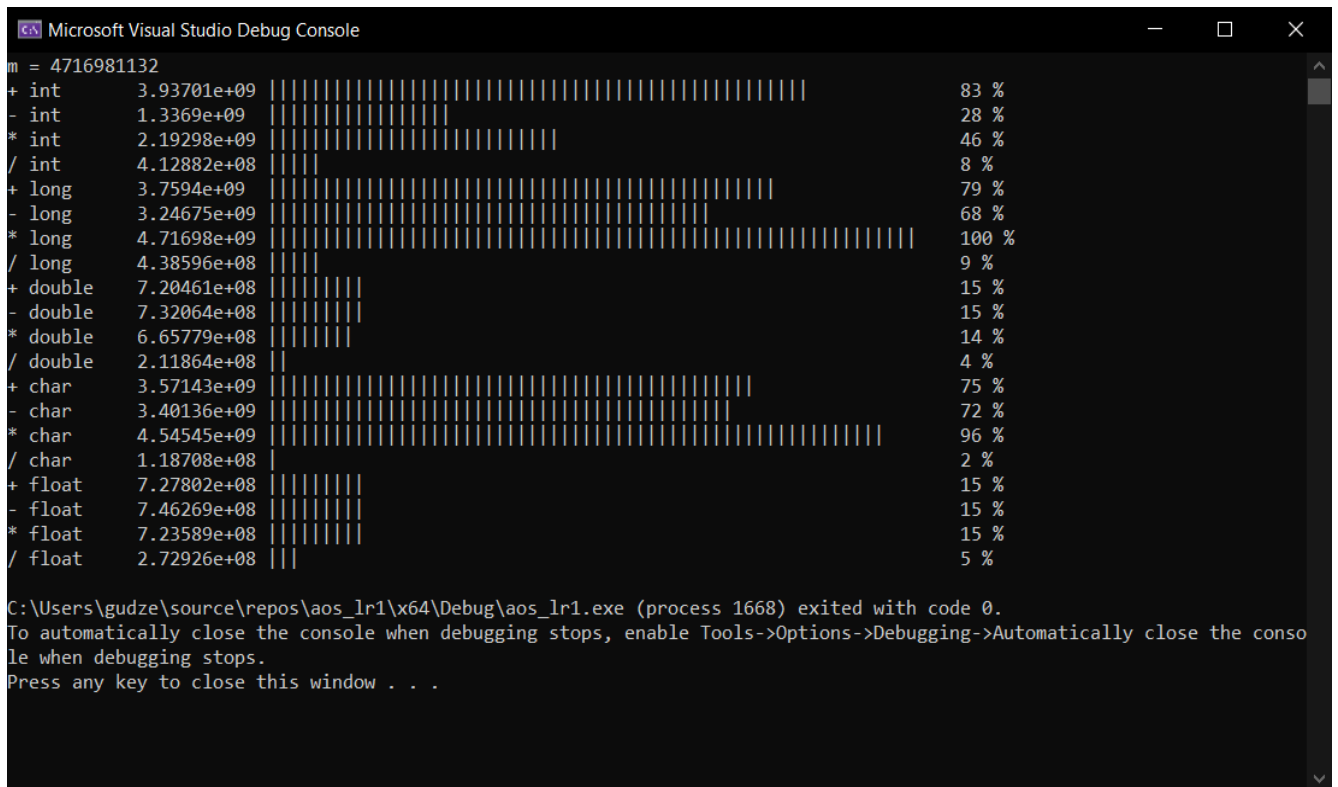
### Запуск на Windows (основній машині)

Операційна система: Windows 10 Pro

Процесор: AMD Ryzen 5 3550H (2.10 ГГц) 64-bit

Оперативна пам'ять: 8,00 ГБ (доступно: 5,88 ГБ).

Під час тестування найкращі показники швидкодії (найбільшу кількість виконуваних операцій за секунду) отримали для операцій додавання та множення типів «int», «float», «double». Найгірші показники мають операції ділення.



```
m = 4716981132
+ int      3.93701e+09 ||| 83 %
- int      1.3369e+09 ||| 28 %
* int      2.19298e+09 ||| 46 %
/ int      4.12882e+08 ||| 8 %
+ long     3.7594e+09 ||| 79 %
- long     3.24675e+09 ||| 68 %
* long     4.71698e+09 ||| 100 %
/ long     4.38596e+08 ||| 9 %
+ double   7.20461e+08 ||| 15 %
- double   7.32064e+08 ||| 15 %
* double   6.65779e+08 ||| 14 %
/ double   2.11864e+08 ||| 4 %
+ char     3.57143e+09 ||| 75 %
- char     3.40136e+09 ||| 72 %
* char     4.54545e+09 ||| 96 %
/ char     1.18708e+08 ||| 2 %
+ float    7.27802e+08 ||| 15 %
- float    7.46269e+08 ||| 15 %
* float    7.23589e+08 ||| 15 %
/ float    2.72926e+08 ||| 5 %

C:\Users\gudze\source\repos\aos_lr1\x64\Debug\aos_lr1.exe (process 1668) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

### Запуск на Android через додаток «Cxxdroid».

Операційна система: Android 11

Процесор: Samsung Exynos 9820 (2.7 ГГц, 2.3 ГГц, 1.9 ГГц), 8-ядерний, 64-bit

Оперативна пам'ять: 8,00 ГБ.

Під час тестування найкращі показники швидкодії (найбільшу кількість виконуваних операцій за секунду) отримали для операцій додавання та множення типів «char», «int», «float». Найгірші показники мають операції ділення.

```
←          TAB  _  :
+   int    4.319286e+08 |||||
||||| 100 %
-   int    4.314569e+08 |||||
||||| 99 %
*   int    2.365633e+08 |||||
||||| 54 %
/   int    1.647708e+08 |||||
||||| 38 %
+   long   4.270175e+08 |||||
||||| 98 %
-   long   4.319106e+08 |||||
||||| 99 %
*   long   2.164314e+08 |||||
||||| 50 %
/   long   1.344649e+08 |||||
||||| 31 %
+   double 1.170363e+08 |||||
||||| 27 %
-   double 1.190227e+08 |||||
||||| 27 %
*   double 1.189219e+08 |||||
||||| 27 %
/   double 9.142217e+07 |||||
||||| 21 %
+   float  1.190343e+08 |||||
||||| 27 %
-   float  1.171362e+08 |||||
||||| 27 %
*   float  1.190939e+08 |||||
||||| 27 %
/   float  9.156780e+07 |||||
||||| 21 %
+   char   2.538585e+08 |||||
||||| 58 %
-   char   2.538252e+08 |||||
||||| 58 %
*   char   1.790172e+08 |||||
||||| 41 %
/   char   1.520451e+08 |||||
||||| 35 %

[Program finished]
```

//

+ int	4.319286e+08		100 %
- int	4.314569e+08		99 %
* int	2.365633e+08		54 %
/ int	1.647708e+08		38 %
+ long	4.270175e+08		98 %
- long	4.319106e+08		99 %
* long	2.164314e+08		50 %
/ long	1.344649e+08		31 %
+ double	1.170363e+08		27 %
- double	1.190227e+08		27 %
* double	1.189219e+08		27 %
/ double	9.142217e+07		21 %
+ float	1.190343e+08		27 %
- float	1.171362e+08		27 %
* float	1.190939e+08		27 %
/ float	9.156780e+07		21 %
+ char	2.538585e+08		58 %
- char	2.538252e+08		58 %
* char	1.790172e+08		41 %
/ char	1.520451e+08		35 %

[Program finished]

//

## Висновок

На Windows для всіх типів операції додавання виконуються або швидше ніж операції віднімання, або з однаковою швидкістю. На Android операції додавання, віднімання та множення для всіх типів виконуються приблизно з однією швидкістю. На обидвох ОС для усіх

типів операції ділення виконуються найповільніше, а операції множення виконуються приблизно з такою ж швидкістю, як і операції додавання. На Windows операції зі змінними типу «char» загалом оброблялись найповільніше, проте такі операції на Android показали найкраще результати. Операції на Android обробляються повільніше ніж на Windows, проте результати стабільніші.

## Додаток. Код програми

[illegible]

```
c = a + b;
c = a + b;
}
stop = clock();
}
else if (operation == '-') {
start = clock();
for (int i = 0; i < TICK_COUNT; i++) {
c = a - b;
c = a - b;
c = a - b;
c = a - b;
c = a - b;
c = a - b;
c = a - b;
c = a - b;
c = a - b;
c = a - b;
}
stop = clock();
}
else if (operation == '*') {
start = clock();
for (int i = 0; i < TICK_COUNT; i++) {
c = a * b;
c = a * b;
c = a * b;
c = a * b;
c = a * b;
c = a * b;
c = a * b;
c = a * b;
}
```



```

c = a * b;
c = a * b;
c = a * b;
}
stop = clock();
}
else {
start = clock();
for (int i = 0; i < TICK_COUNT; i++) {
c = a / b;
c = a / b;
c = a / b;
c = a / b;
c = a / b;
c = a / b;
c = a / b;
c = a / b;
c = a / b;
c = a / b;
}
stop = clock();
}

clock_t loop_start = clock();
for (int i = 0; i < TICK_COUNT; i++) {

}
clock_t loop_stop = clock();
return (double)(stop - start + loop_start - loop_stop) / CLK_TCK;
}

```

```

long long tick_per_second(double time) {
return (long long)((TICK_COUNT * 10) / time);
}

```

```

////////////////////////////////////

```

```

void Diag(int height, long long max_value, long long value) {
int s_count = (int)((double)value / (double)max_value * height);
for (int i = 0; i < s_count; i++) {
cout << "|";
}
for (int j = 0; j < height - s_count; j++) {
cout << " ";
}
cout << "\t" << (int)((double)value / (double)max_value * 100) << " %\n";
}

```

```

void Statist(vector<pair<string, long long>> vect) {
int diag_height = 60;
long long m = -1;
for (int i = 0; i < vect.size(); i++) {
if (vect[i].second > m) {
m = vect[i].second;
}
}
cout << "m = " << m << endl;
for (const auto& pair : vect) {
cout << pair.first;
for (int i = 0; i < (12 - pair.first.size()); i++) {
cout << " ";
}
}
}

```

```
cout << (double)pair.second << "\t";
```

```
Diag(diag_height, m, pair.second);
```

```
}
```

```
}
```

```
int main() {
```

```
vector <pair <string, long long>> res;
```

```
int a1 = 24343, b1 = 51343;
```

```
long a2 = 24343, b2 = 51343;
```

```
double a3 = 424.54535345, b3 = 342.54535345;
```

```
long long a4 = 2434343242, b4 = 513433423432;
```

```
float a5 = 424.2332, b5 = 342.5232;
```

```
string operations = "+-*/";
```

```
for (int i = 0; i < 4; i++) {
```

```
res.push_back(make_pair(string(1, operations[i]) + " int", tick_per_second(Time(operations[i], a1, b1))));
```

```
}
```

```
for (int i = 0; i < 4; i++) {
```

```
res.push_back(make_pair(string(1, operations[i]) + " long", tick_per_second(Time(operations[i], a2, b2))));
```

```
}
```

```
for (int i = 0; i < 4; i++) {
```

```
res.push_back(make_pair(string(1, operations[i]) + " double", tick_per_second(Time(operations[i], a3, b3))));
```

```
}
```

```
for (int i = 0; i < 4; i++) {
```

```
res.push_back(make_pair(string(1, operations[i]) + " char", tick_per_second(Time(operations[i], a4, b4))));
```

```
}
```

```
for (int i = 0; i < 4; i++) {
```

```
res.push_back(make_pair(string(1, operations[i]) + " float", tick_per_second(Time(operations[i], a5, b5))));
```

```
}  
Statist(res);  
return 0;  
}
```