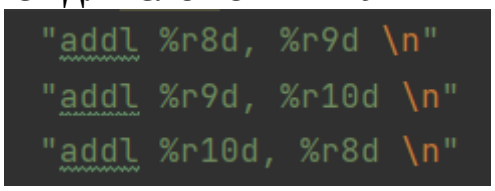
1.Определите среднее время работы простых арифметических операций

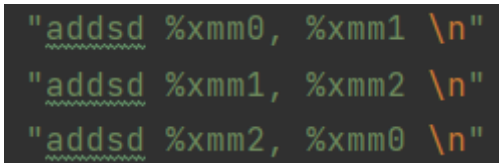
for (int i = 0; i <1,000,000,000; ++ i) по 200 сложений
Без операций время работы равно 1.245838440000000131340e-08
Это будет абсолютной погрешностью
С операциями среднее время работы 200 сложений составляет
5.3742082299996271573 5e-08±1.245838440000000131340e-08

Вот такой код использовался для сложений int



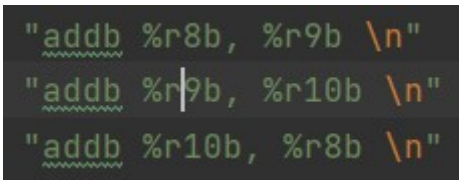
Значит время работы одной операции add для int составляет:
$t = (2.7 \pm 0.6)*10^{-10}$ с
У меня тактовая частота процессора 1.6 ГГц. Соответственно один так должен совершаться за $1/1.6*10^{9} = 6.25*10^{-10}$ с. Ну да примерно так и получается в пределах погрешности полученное экспериментально значение совпадает.

Код для double



С операциями среднее время работы 200 сложений составляет
5.233349959999833565 58e-08±1.217449060000003928 29e-08
Значит время работы одной операции add для double составляет:
$t = (2.6 \pm 0.6)*10^{-10}$ с

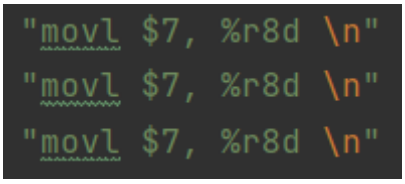
Код для char



С операциями среднее время работы 200 сложений составляет
5.453514499999574650 18e-08±1.191231119999998941 07e-08
Значит время работы одной операции add для char составляет:
$t = (2.7 \pm 0.6)*10^{-10}$ с

Среднее время записи в ячейку памяти

И таких 200 строк



Получилось время
2.8176859400000 5025628e-08±1.2046787800000 0120410e-08
Значит время работы одной операции mov составляет:
$t = (1.4 \pm 0.6)*10^{-10}$ с

Время работы new, malloc и delete,free

```
for(int i = 0; i<1000; ++i)
{
    auto start = chrono::high_resolution_clock::now();

    int* a = new int[1000000000];

    auto end = chrono::high_resolution_clock::now();
    delete[] a;
    chrono::duration<double> diff = end - start;
    time += diff.count();
}
```

Код на картинке сверху неправильный, так как, по мнению Жени, new работает с [1] и [1000000000] за одинаковое, так как там просто вычисляется сколько байтов нужно и выделяется память одной командой, две константные операции. Код на картинке ниже в листинге выглядит как присваивание в глобальную переменную и вызов функции new 200 раз. Я считаю, что это присваивание **нужно** учесть так как задача в том, чтобы найти сколько времени работает new, а без присваивания new не имеет смыла.

```
for(int i = 0; i<1000000000; ++i)
{
    int* a = new int;
    auto start = chrono::high_resolution_clock::now();

    a = new int;
    a = new int;
    a = new int;
    a = new int;
    a = new int;
    a = new int;
    a = new int;
    a = new int;
    a = new int;
    a = new int;
```

Выполняем 200 операций за один ход цикла 1 миллиард циклов.
Как обычно посмотрим сколько он будет выполнять, когда кода нет.
Без операций время работы равно 1.2458384400000 0131340e-08
Заметка. С этого момента делаю работу дальше после выстрела в свое колено(произошел reboot)
Время работы с операциями, как не сложно догадаться, я не получил. :)
Видимо он не освобождает память при таком переприсваивании. Тогда у меня больше нет идей, буду засекать их вместе.

```
cout << scientific;
cout.precision( prec: 20);
double time = 0;
//cout<<c<<"\n";
   for(int i = 0; i<1000000000; ++i)
   {
       auto start = chrono::high_resolution_clock::now();

       int* a = new int;
       delete a;
       a = new int;
       delete a;
       a = new int;
       delete a;
```

Среднее время работы 100 пар new/delete:
6.209294329999917381 57e-06±1.245838440000001313 40e-08
Среднее время работы пары new/delete:
$t = (6.21 \pm 0.01) * 10^{-8}$ с

```
for(int i = 0; i<10000000; ++i)
{
    auto start = chrono::high_resolution_clock::now();

    unique_ptr<int> data = unique_ptr<int>( p: new int);
    data = unique_ptr<int>( p: new int);
    data = unique_ptr<int>( p: new int);
    data = unique_ptr<int>( p: new int);
```

Среднее время работы 200 unique_ptr:
3.289508131000082216 50e-05±1.245838440000001313 40e-08
Среднее время создания и удаления unique_ptr:
$t = (16.44 \pm 0.01) * 10^{-8}$ с

В 2.5 раза медленнее, чем new/delete

Аналогично:
Среднее время работы 200 shared_ptr:
3.974798809999744369 58e-05±1.245838440000001313 40e-08
Среднее время создания и удаления shared_ptr:
$t = (19.87 \pm 0.01) * 10^{-8}$ с

В 3 раза медленнее, чем new/delete

Определите среднее время разыменования для обычных указателей shared_ptr and unique_ptr.

```
movq    a(%rip), %rax
movl    $1777, (%rax)
```

Разыменование заключается в первой строчке, поэтому по скорости не отличается от mov.
Для unique_ptr вместо movq две строчки leaq и call

```
leaq    a(%rip), %rdi
call    _ZNKSt10unique_ptrIiSt14default_deleteIiEEdeEv
movl    $1777, (%rax)
```

```
for(int i = 0; i<10000000; ++i)
{
    unique_ptr<int> data = unique_ptr<int>( p: new int);
    auto start = chrono::high_resolution_clock::now();
    *data = 1337;
    *data = 1337;
    *data = 1337;
    *data = 1337;
```

Вот этим кодом будем засекать время и потом вычтем из полученного среднее время работы mov
Время работы одной операции mov составляет $t = (1.4 \pm 0.6)*10^{-10}$ с

Среднее время работы разыменования 200 unique_ptr:
2.189396372999960218 53e-06±1.352393049999991584 10e-08
Среднее время разыменования одного unique_ptr:
$t = (1.08\pm0.01)*10^{-8}$ с

Среднее время работы разыменования 200 shared_ptr:
0.940926690000002441065e-06±1.352393049999991584 10e-08
Среднее время разыменования одного shared_ptr:
$t = (0.46\pm0.01)*10^{-8}$ с

Разыменование shared_ptr быстрее в 2.5 раза, чем разыменование unique_ptr.
Разыменование unique_ptr медленнее в 77 раза чем обычная *.
Разыменование shared_ptr медленнее в 32 раза чем обычная *.