HFT

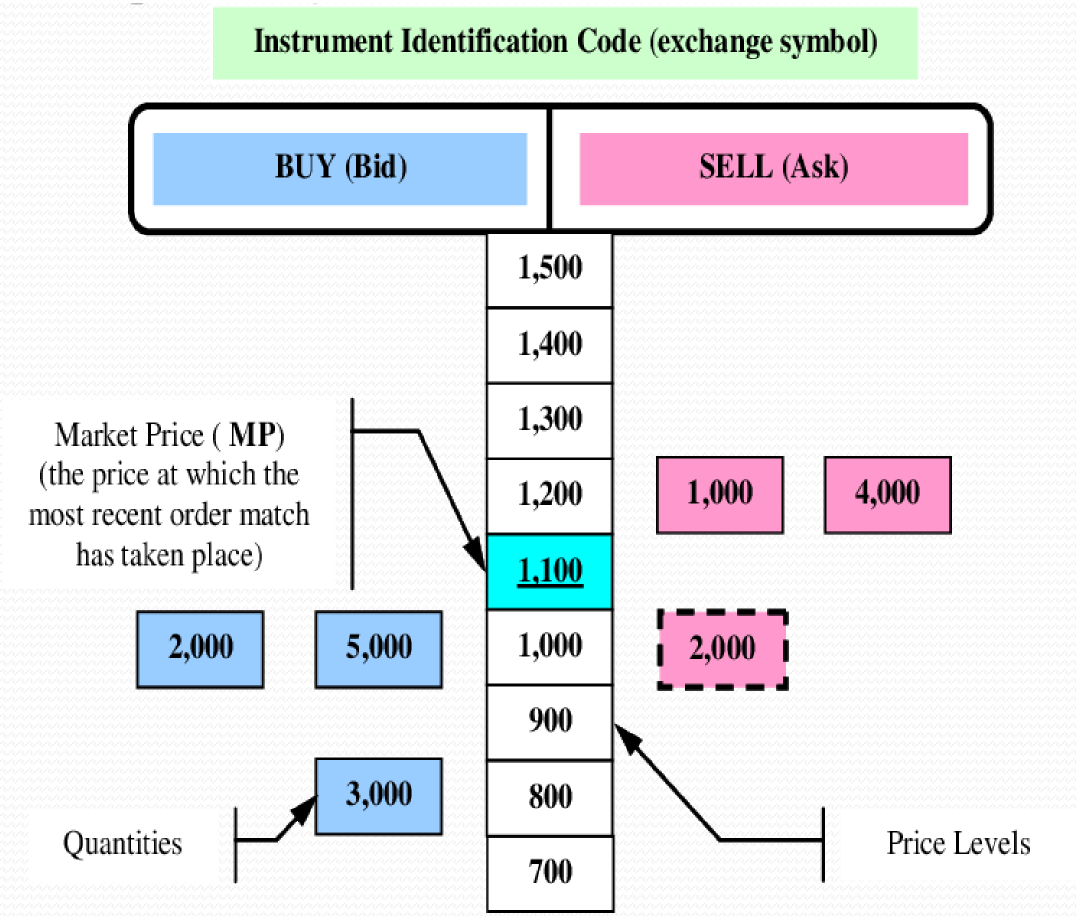
The Old days

- Trading floor – CBOT 2006

Today

- Trading BlackBerry stock - 2013

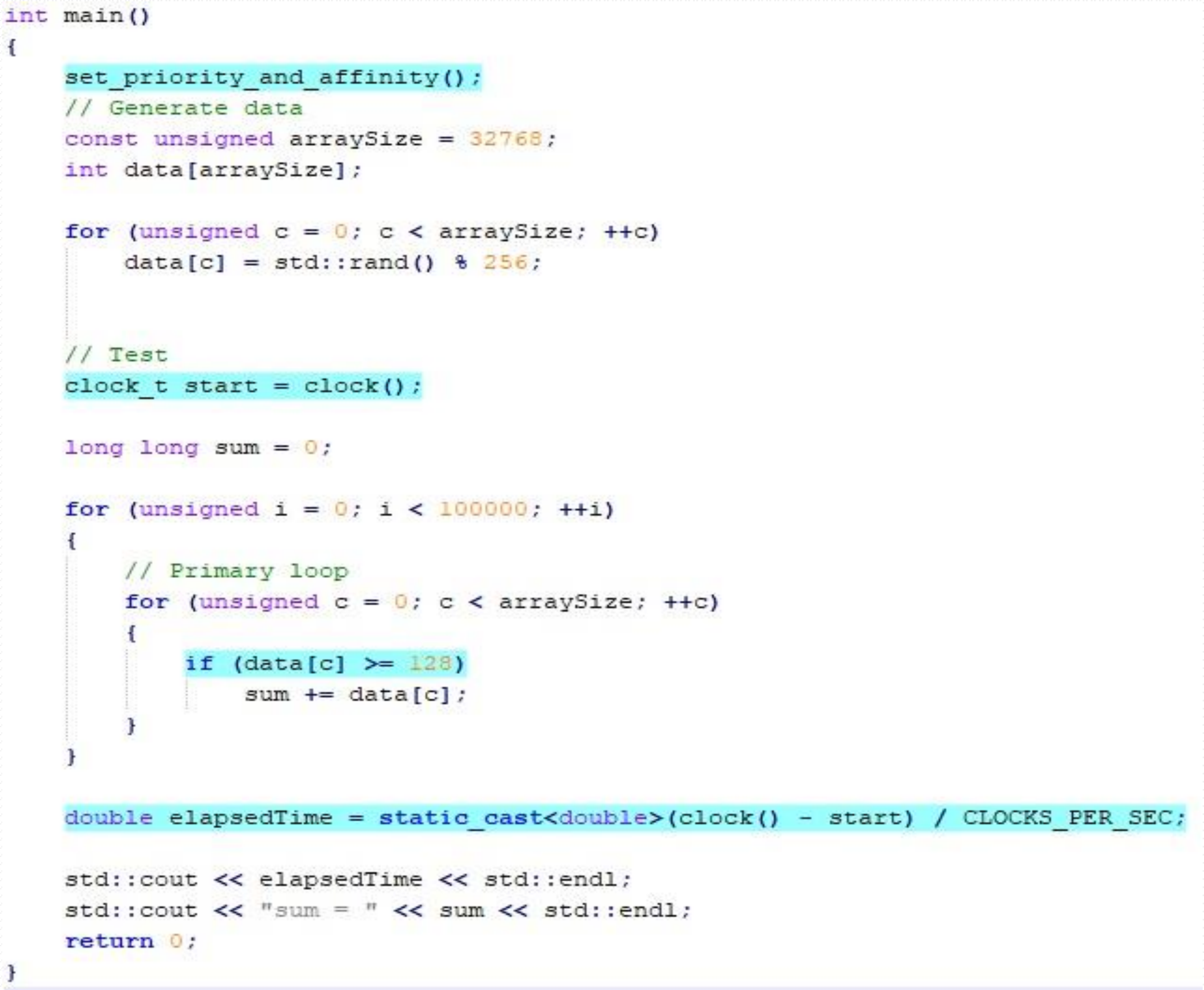
Basic Order flow


Instrument Identification Code (exchange symbol)
BUY (Bid)
SELL (Ask)
1,500
1,400
1,300
Market Price (MP) (the price at which the most recent order match has taken place)
1,200
1,000
4,000
1,100
2,000
5,000
1,000
2,000
900
3,000
800
Quantities
Price Levels
700

How to be fast

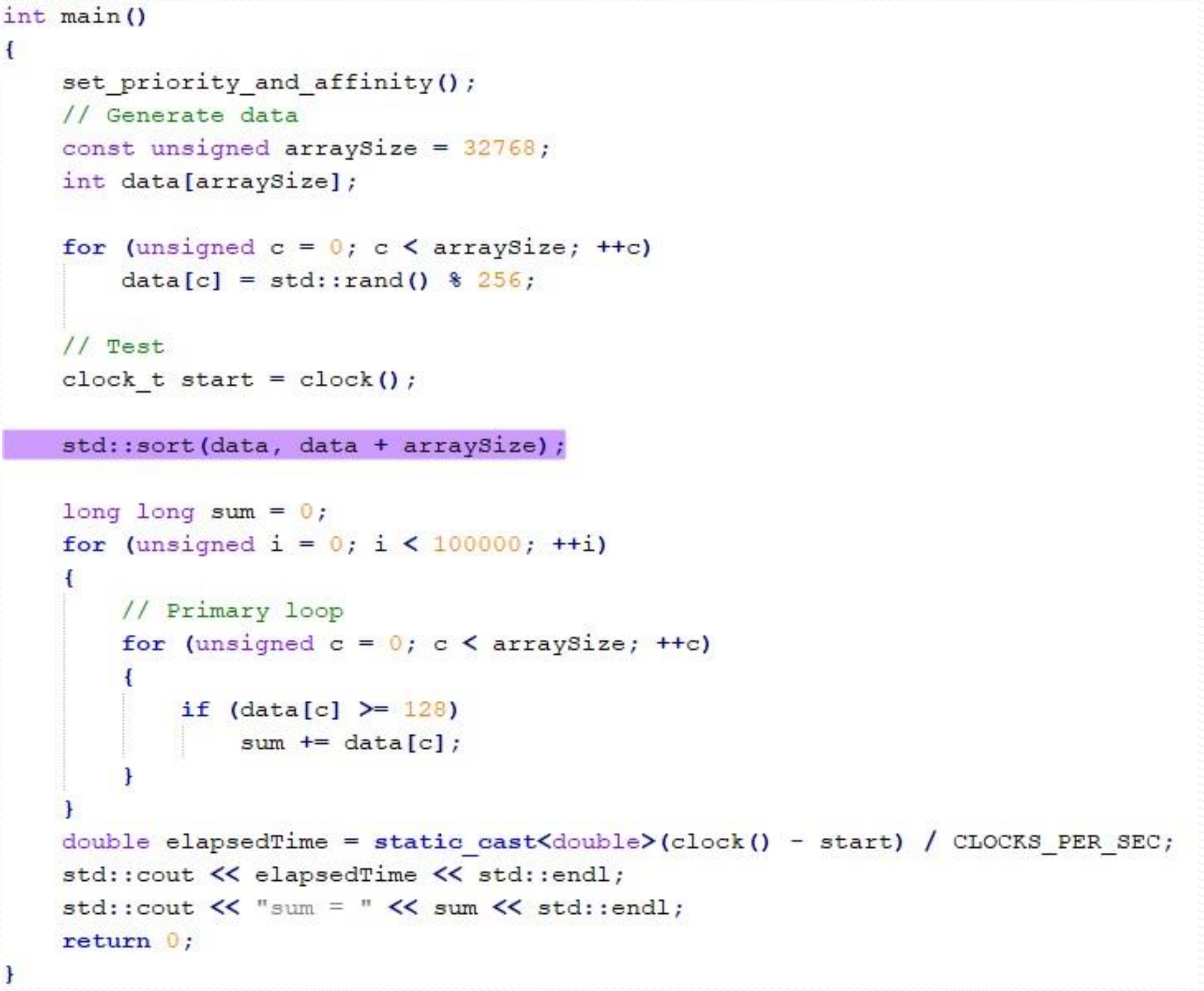
- Physical location
- Know your network
- Know your OS
- Measure, Measure, Measure
- Write fast code

Write fast code

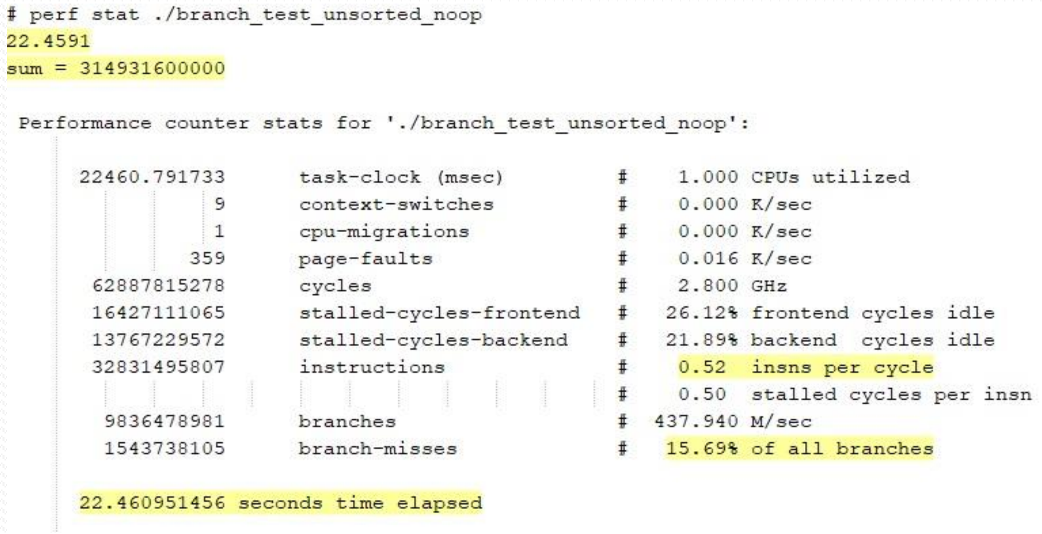


```
int main()
{
    set_priority_and_affinity();
    // Generate data
    const unsigned arraySize = 32768;
    int data[arraySize];

    for (unsigned c = 0; c < arraySize; ++c)
        data[c] = std::rand() % 256;


    // Test
    clock_t start = clock();

    long long sum = 0;

    for (unsigned i = 0; i < 100000; ++i)
    {
        // Primary loop
        for (unsigned c = 0; c < arraySize; ++c)
        {
            if (data[c] >= 128)
                sum += data[c];
        }
    }

    double elapsedTime = static_cast<double>(clock() - start) / CLOCKS_PER_SEC;

    std::cout << elapsedTime << std::endl;
    std::cout << "sum = " << sum << std::endl;
    return 0;
}
```

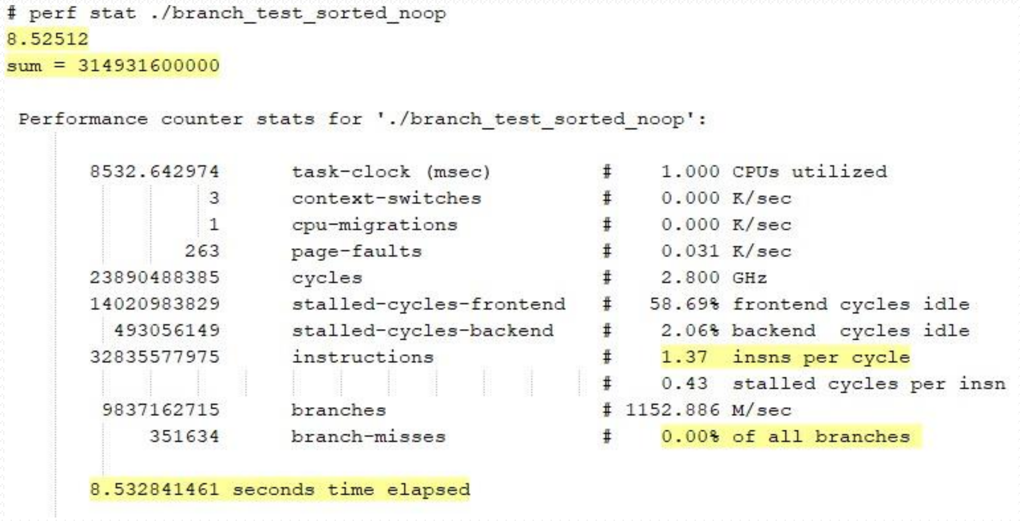
Write fast code



```
int main()
{
    set_priority_and_affinity();
    // Generate data
    const unsigned arraySize = 32768;
    int data[arraySize];

    for (unsigned c = 0; c < arraySize; ++c)
        data[c] = std::rand() % 256;

    // Test
    clock_t start = clock();

    std::sort(data, data + arraySize);

    long long sum = 0;
    for (unsigned i = 0; i < 100000; ++i)
    {
        // Primary loop
        for (unsigned c = 0; c < arraySize; ++c)
        {
            if (data[c] >= 128)
                sum += data[c];
        }
    }
    double elapsedTime = static_cast<double>(clock() - start) / CLOCKS_PER_SEC;
    std::cout << elapsedTime << std::endl;
    std::cout << "sum = " << sum << std::endl;
    return 0;
}
```

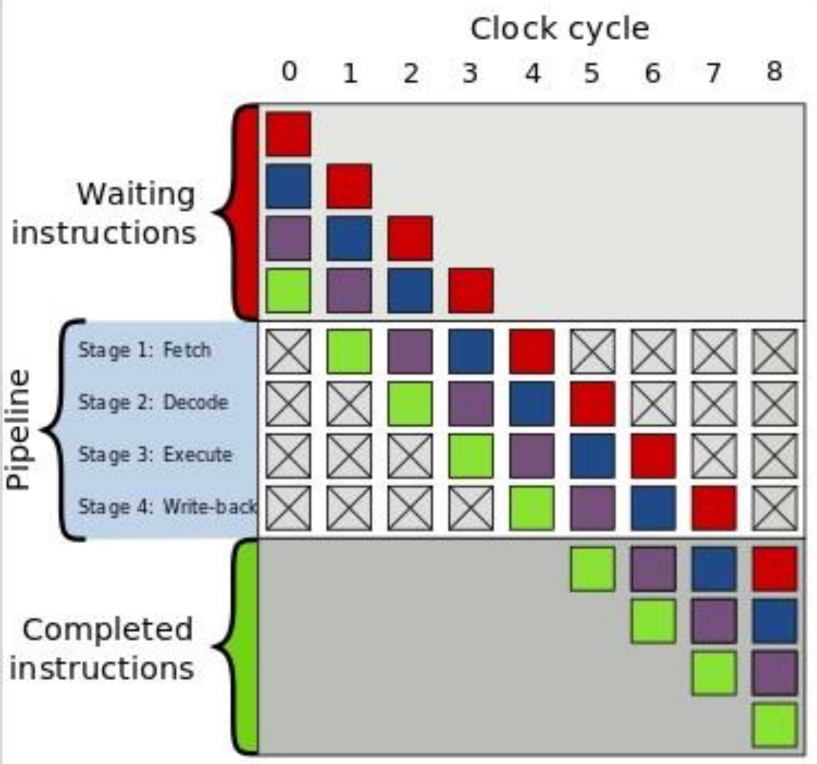


```
# perf stat ./branch_test_unsorted_noop
22.4591
sum = 314931600000

 Performance counter stats for './branch_test_unsorted_noop':

     22460.791733      task-clock (msec)         #    1.000 CPUs utilized
                9      context-switches          #    0.000 K/sec
                1      cpu-migrations            #    0.000 K/sec
              359      page-faults               #    0.016 K/sec
      62887815278      cycles                    #    2.800 GHz
      16427111065      stalled-cycles-frontend   #   26.12% frontend cycles idle
      13767229572      stalled-cycles-backend    #   21.89% backend  cycles idle
      32831495807      instructions              #    0.52  insns per cycle
                                                 #    0.50  stalled cycles per insn
       9836478981      branches                  #  437.940 M/sec
       1543738105      branch-misses             #   15.69% of all branches

     22.460951456 seconds time elapsed
```

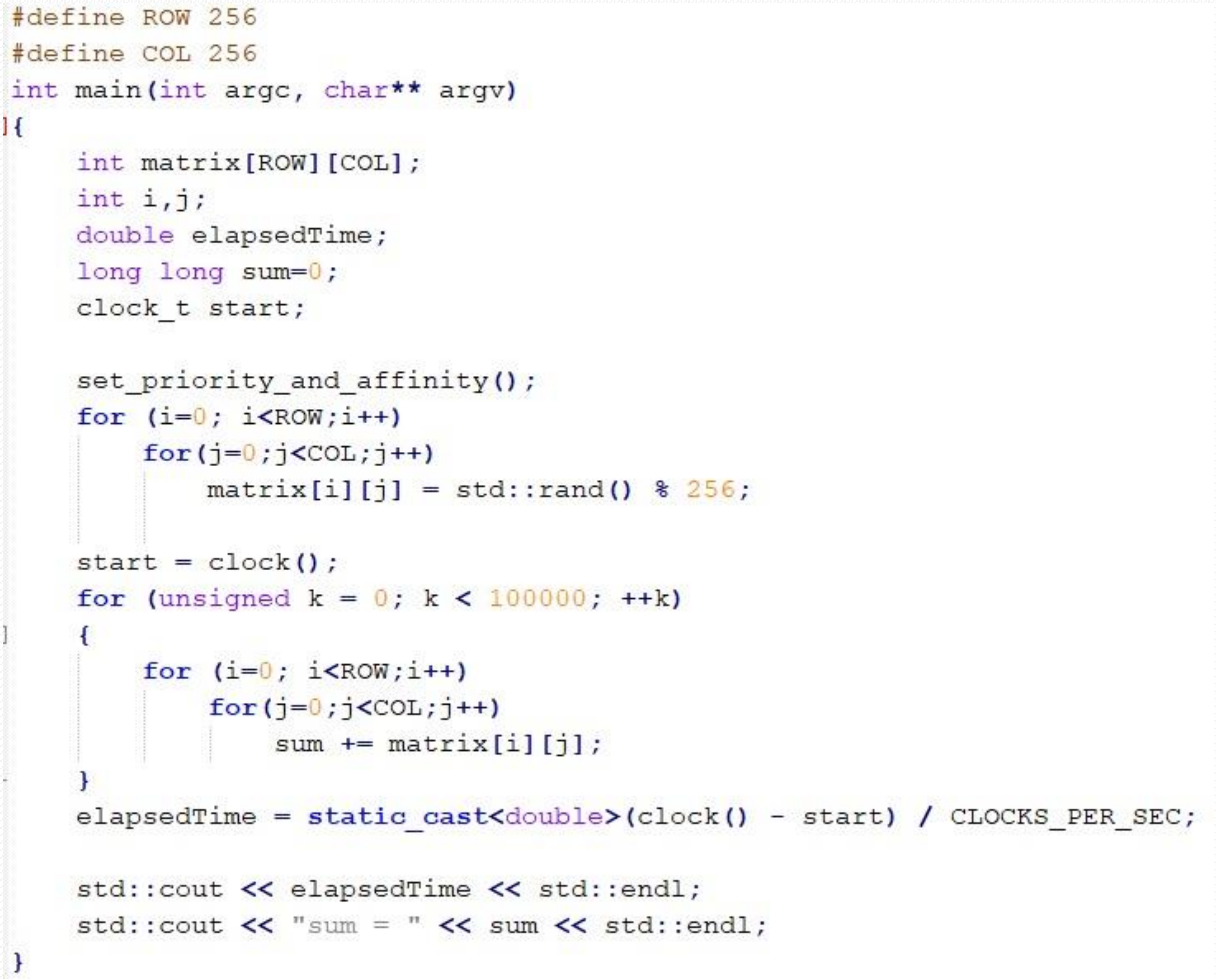


```
# perf stat ./branch_test_sorted_noop
8.52512
sum = 314931600000

 Performance counter stats for './branch_test_sorted_noop':

      8532.642974      task-clock (msec)         #    1.000 CPUs utilized
                3      context-switches          #    0.000 K/sec
                1      cpu-migrations            #    0.000 K/sec
              263      page-faults               #    0.031 K/sec
      23890488385      cycles                    #    2.800 GHz
      14020983829      stalled-cycles-frontend   #   58.69% frontend cycles idle
        493056149      stalled-cycles-backend    #    2.06% backend  cycles idle
      32835577975      instructions              #    1.37  insns per cycle
                                                 #    0.43  stalled cycles per insn
       9837162715      branches                  # 1152.886 M/sec
           351634      branch-misses             #    0.00% of all branches

      8.532841461 seconds time elapsed
```

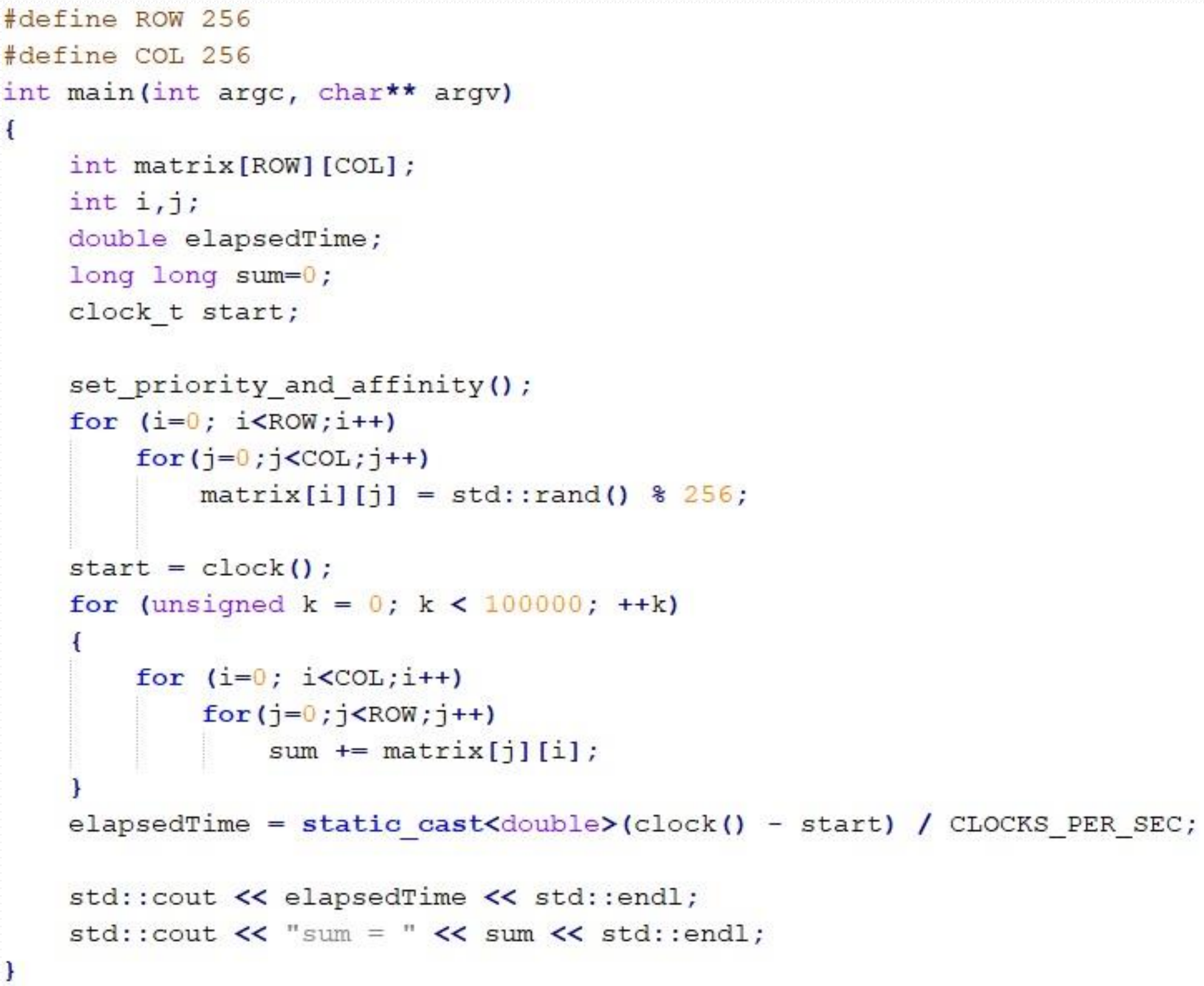
CPU pipeline



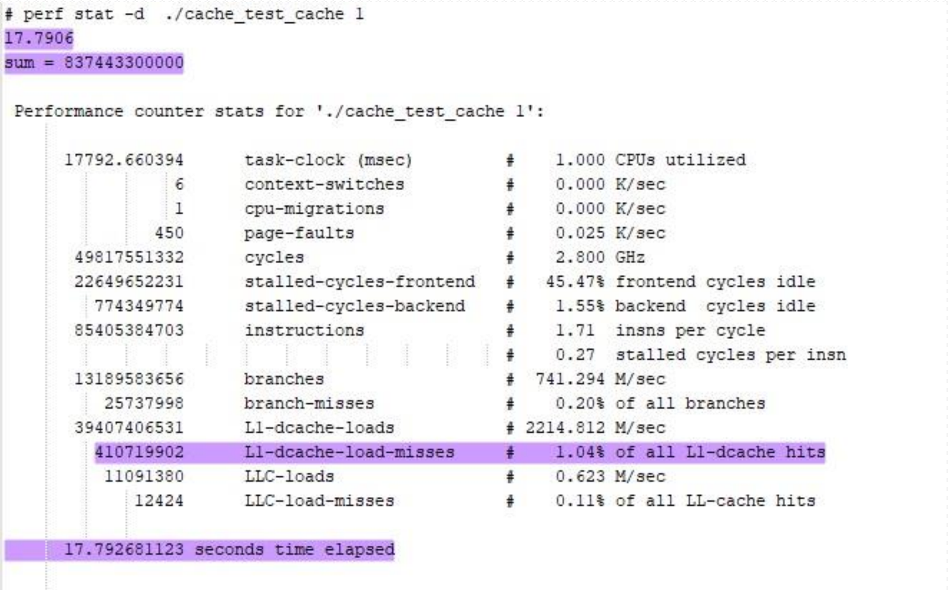
Generic 4-stage pipeline; the colored boxes represent instructions independent of each other



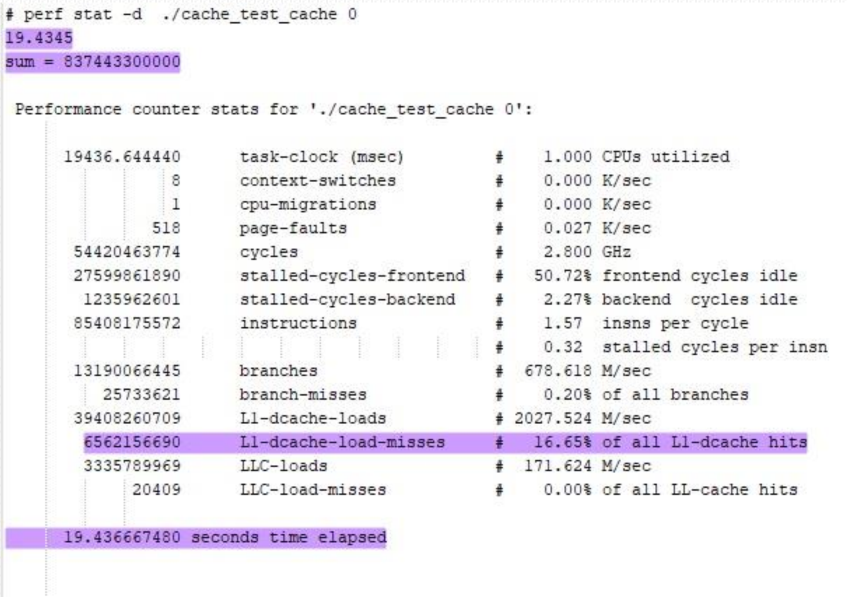
```cpp
#define ROW 256
#define COL 256
int main(int argc, char** argv)
{
    int matrix[ROW][COL];
    int i,j;
    double elapsedTime;
    long long sum=0;
    clock_t start;

    set_priority_and_affinity();
    for (i=0; i<ROW;i++)
        for(j=0;j<COL;j++)
            matrix[i][j] = std::rand() % 256;

    start = clock();
    for (unsigned k = 0; k < 100000; ++k)
    {
        for (i=0; i<ROW;i++)
            for(j=0;j<COL;j++)
                sum += matrix[i][j];
    }
    elapsedTime = static_cast<double>(clock() - start) / CLOCKS_PER_SEC;

    std::cout << elapsedTime << std::endl;
    std::cout << "sum = " << sum << std::endl;
}
```



```
#define ROW 256
#define COL 256
int main(int argc, char** argv)
{
    int matrix[ROW][COL];
    int i,j;
    double elapsedTime;
    long long sum=0;
    clock_t start;

    set_priority_and_affinity();
    for (i=0; i<ROW;i++)
        for(j=0;j<COL;j++)
            matrix[i][j] = std::rand() % 256;

    start = clock();
    for (unsigned k = 0; k < 100000; ++k)
    {
        for (i=0; i<COL;i++)
            for(j=0;j<ROW;j++)
                sum += matrix[j][i];
    }
    elapsedTime = static_cast<double>(clock() - start) / CLOCKS_PER_SEC;

    std::cout << elapsedTime << std::endl;
    std::cout << "sum = " << sum << std::endl;
}
```



```
# perf stat -d  ./cache_test_cache 1
17.7906
sum = 837443300000

 Performance counter stats for './cache_test_cache 1':

      17792.660394      task-clock (msec)         #    1.000 CPUs utilized
                 6      context-switches          #    0.000 K/sec
                 1      cpu-migrations            #    0.000 K/sec
               450      page-faults               #    0.025 K/sec
       49817551332      cycles                    #    2.800 GHz
       22649652231      stalled-cycles-frontend   #   45.47% frontend cycles idle
         774349774      stalled-cycles-backend    #    1.55% backend  cycles idle
       85405384703      instructions              #    1.71  insns per cycle
                                                  #    0.27  stalled cycles per insn
       13189583656      branches                  #  741.294 M/sec
          25737998      branch-misses             #    0.20% of all branches
       39407406531      L1-dcache-loads           # 2214.812 M/sec
         410719902      L1-dcache-load-misses     #    1.04% of all L1-dcache hits
          11091380      LLC-loads                 #    0.623 M/sec
             12424      LLC-load-misses           #    0.11% of all LL-cache hits

      17.792681123 seconds time elapsed
```

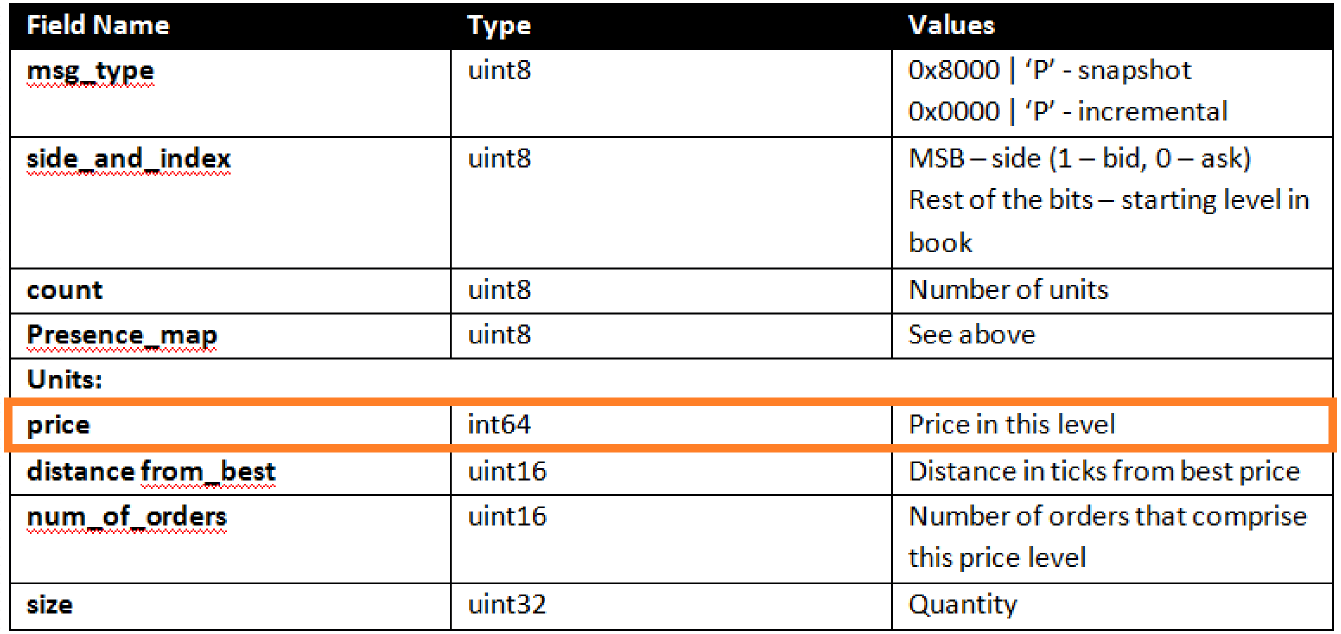


```
# perf stat -d  ./cache_test_cache 0
19.4345
sum = 837443300000

 Performance counter stats for './cache_test_cache 0':

      19436.644440      task-clock (msec)         #    1.000 CPUs utilized
                 8      context-switches          #    0.000 K/sec
                 1      cpu-migrations            #    0.000 K/sec
               518      page-faults               #    0.027 K/sec
       54420463774      cycles                    #    2.800 GHz
       27599861890      stalled-cycles-frontend   #   50.72% frontend cycles idle
        1235962601      stalled-cycles-backend    #    2.27% backend  cycles idle
       85408175572      instructions              #    1.57  insns per cycle
                                                  #    0.32  stalled cycles per insn
       13190066445      branches                  #  678.618 M/sec
          25733621      branch-misses             #    0.20% of all branches
       39408260709      L1-dcache-loads           # 2027.524 M/sec
        6562156690      L1-dcache-load-misses     #   16.65% of all L1-dcache hits
        3335789969      LLC-loads                 #  171.624 M/sec
             20409      LLC-load-misses           #    0.00% of all LL-cache hits

      19.436667480 seconds time elapsed
```

Code challenge

Write fast code – real example

- MD prices are published by the exchange as scaled integer, we need to generate code which changes the exponent as fast as possible



Field Name	Type	Values
msg_type	uint8	0x8000 | 'P' - snapshot 0x0000 | 'P' - incremental
side_and_index	uint8	MSB – side (1 – bid, 0 – ask) Rest of the bits – starting level in book
count	uint8	Number of units
Presence_map	uint8	See above
Units:		
price	int64	Price in this level
distance from_best	uint16	Distance in ticks from best price
num_of_orders	uint16	Number of orders that comprise this price level
size	uint32	Quantity

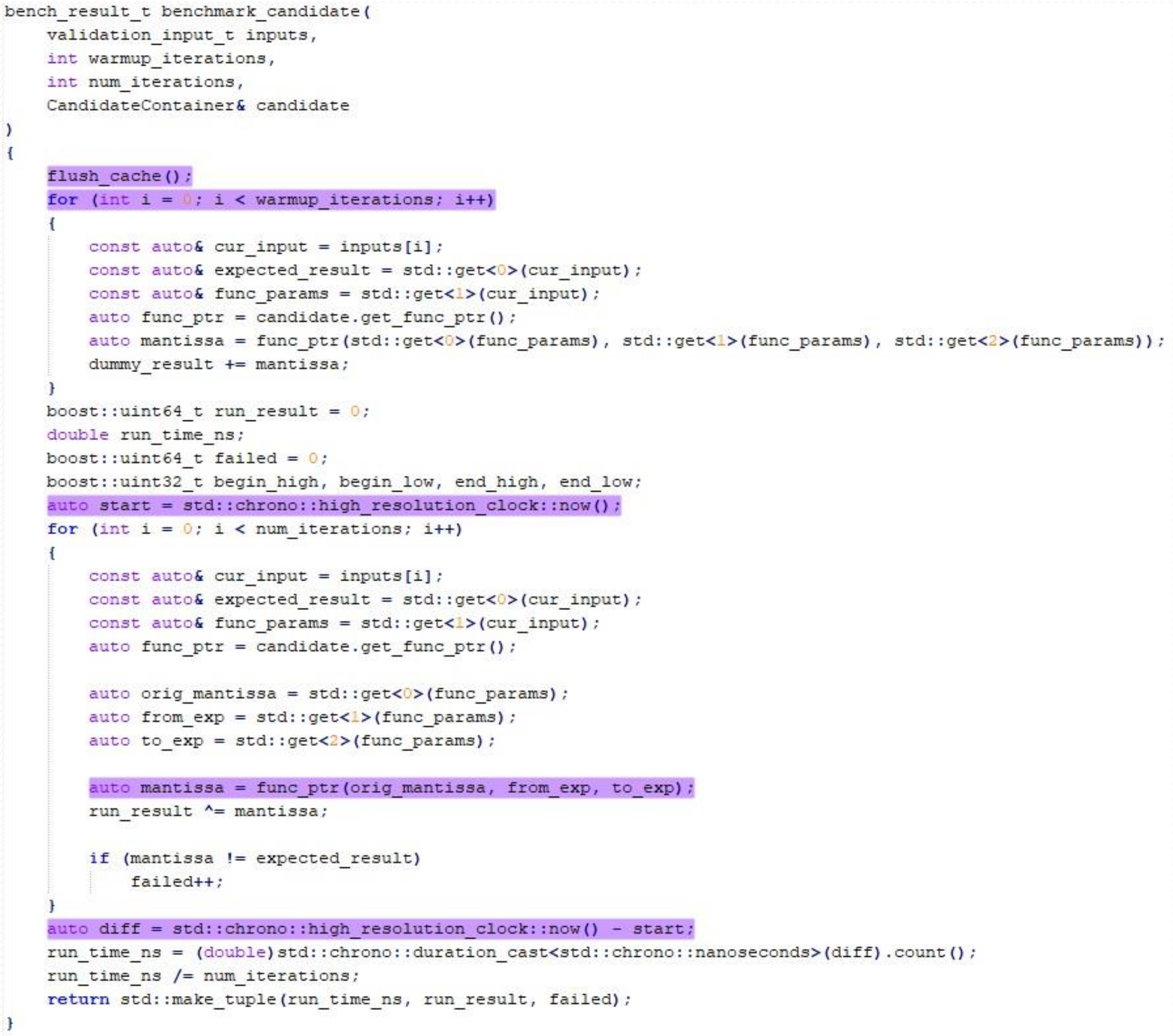
Write fast code – real example

True Value	Equalsto …	Equals to…
1,500	1500 *10^{0}	15 *10^{2}
1,500,000,000	150000 *10^{4}	1500 *10^{6}
0.15	15 *10^{-2}	1500 *10^{-4}

Base Version (66ns)

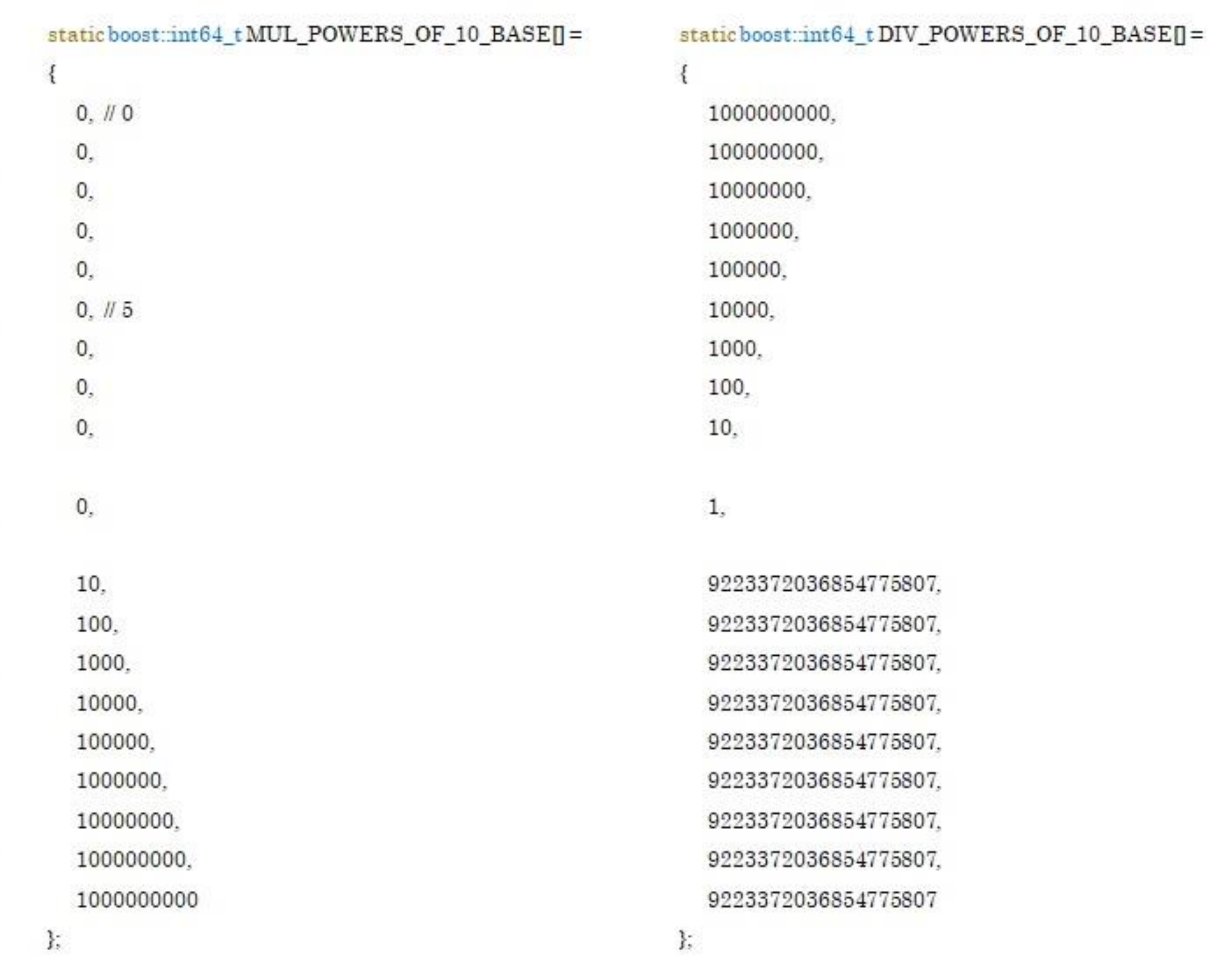
```
boost::int64_t changeExponent(boost::int64_t mantissa, boost::int16_t from_exp,
                              boost::int16_t to_exp)
{
    boost::int16_t diff = to_exp - from_exp;
    if (!diff)
        return mantissa;
    if (diff <= 0)
    {
        boost::int64_t div = (boost::int64_t)pow(10.0,(double)abs(diff));
        return mantissa * div;
    }
    else
    {
        boost::uint32_t div = (boost::uint32_t)pow(10.0,(double)abs(diff));
        return mantissa / div;
    }
}
```

Benchmark utility



```
bench_result_t benchmark_candidate(
    validation_input_t inputs,
    int warmup_iterations,
    int num_iterations,
    CandidateContainer& candidate
)
{
    flush_cache();
    for (int i = 0; i < warmup_iterations; i++)
    {
        const auto& cur_input = inputs[i];
        const auto& expected_result = std::get<0>(cur_input);
        const auto& func_params = std::get<1>(cur_input);
        auto func_ptr = candidate.get_func_ptr();
        auto mantissa = func_ptr(std::get<0>(func_params), std::get<1>(func_params), std::get<2>(func_params));
        dummy_result += mantissa;
    }
    boost::uint64_t run_result = 0;
    double run_time_ns;
    boost::uint64_t failed = 0;
    boost::uint32_t begin_high, begin_low, end_high, end_low;
    auto start = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < num_iterations; i++)
    {
        const auto& cur_input = inputs[i];
        const auto& expected_result = std::get<0>(cur_input);
        const auto& func_params = std::get<1>(cur_input);
        auto func_ptr = candidate.get_func_ptr();

        auto orig_mantissa = std::get<0>(func_params);
        auto from_exp = std::get<1>(func_params);
        auto to_exp = std::get<2>(func_params);

        auto mantissa = func_ptr(orig_mantissa, from_exp, to_exp);
        run_result ^= mantissa;

        if (mantissa != expected_result)
            failed++;
    }
    auto diff = std::chrono::high_resolution_clock::now() - start;
    run_time_ns = (double)std::chrono::duration_cast<std::chrono::nanoseconds>(diff).count();
    run_time_ns /= num_iterations;
    return std::make_tuple(run_time_ns, run_result, failed);
}
```

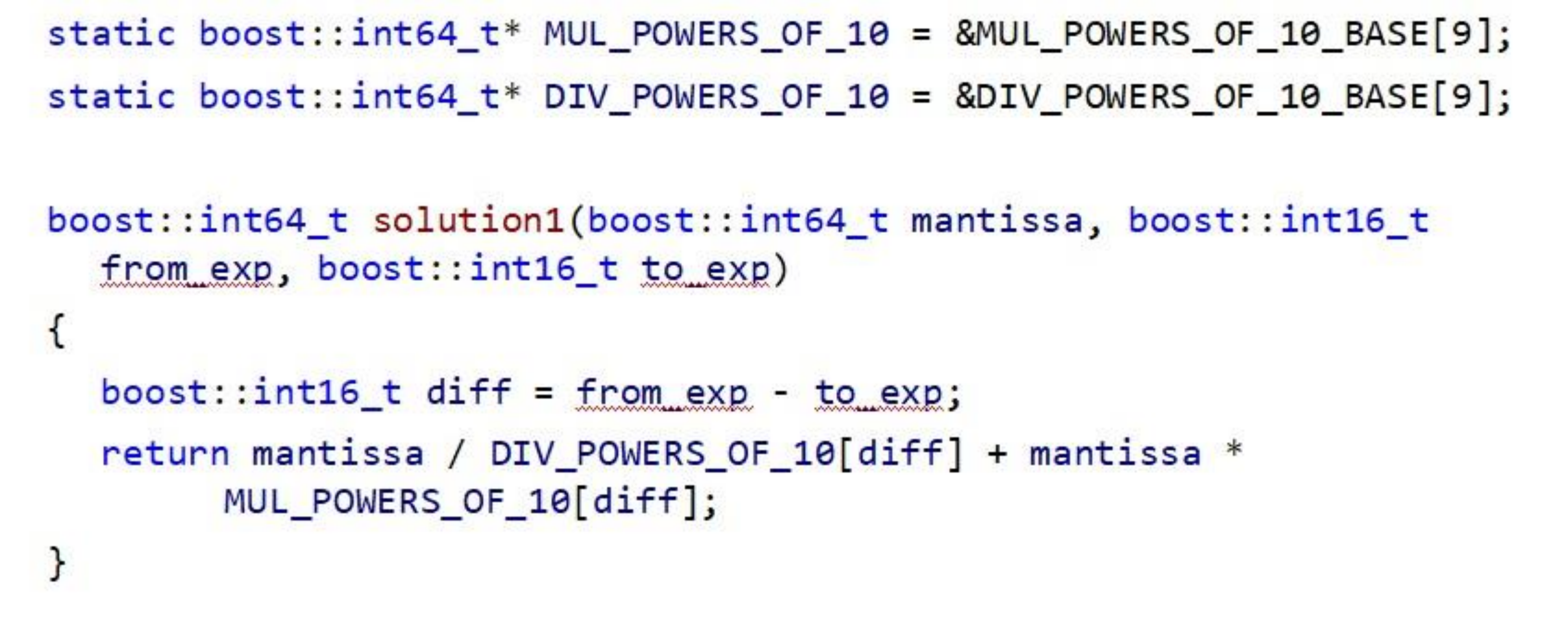
Solution 1 :



```
static boost::int64_t MUL_POWERS_OF_10_BASE[] =
{
    0, // 0
    0,
    0,
    0,
    0,
    0, // 5
    0,
    0,
    0,

    0,

    10,
    100,
    1000,
    10000,
    100000,
    1000000,
    10000000,
    100000000,
    1000000000
};
```

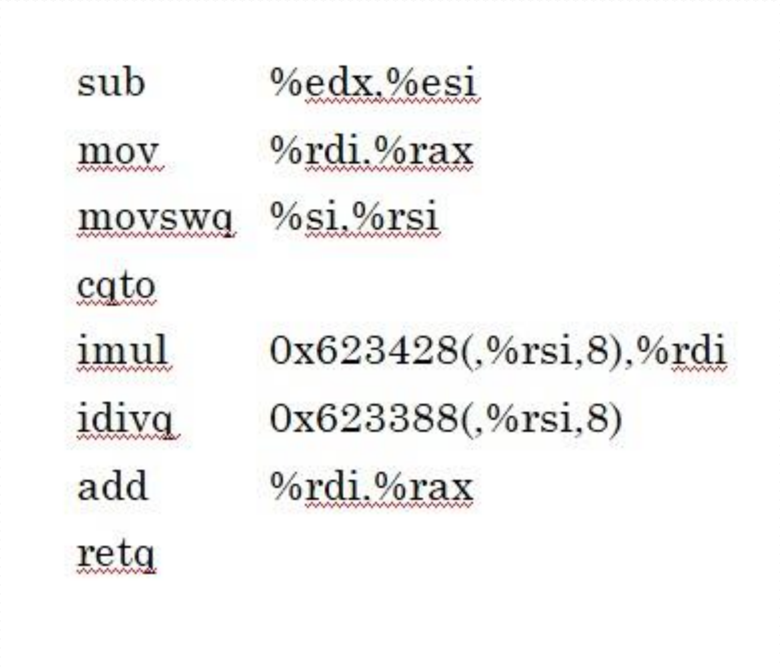
```
static boost::int64_t DIV_POWERS_OF_10_BASE[] =
{
    1000000000,
    100000000,
    10000000,
    1000000,
    100000,
    10000,
    1000,
    100,
    10,

    1,

    9223372036854775807,
    9223372036854775807,
    9223372036854775807,
    9223372036854775807,
    9223372036854775807,
    9223372036854775807,
    9223372036854775807,
    9223372036854775807,
    9223372036854775807
};
```

Solution 1:



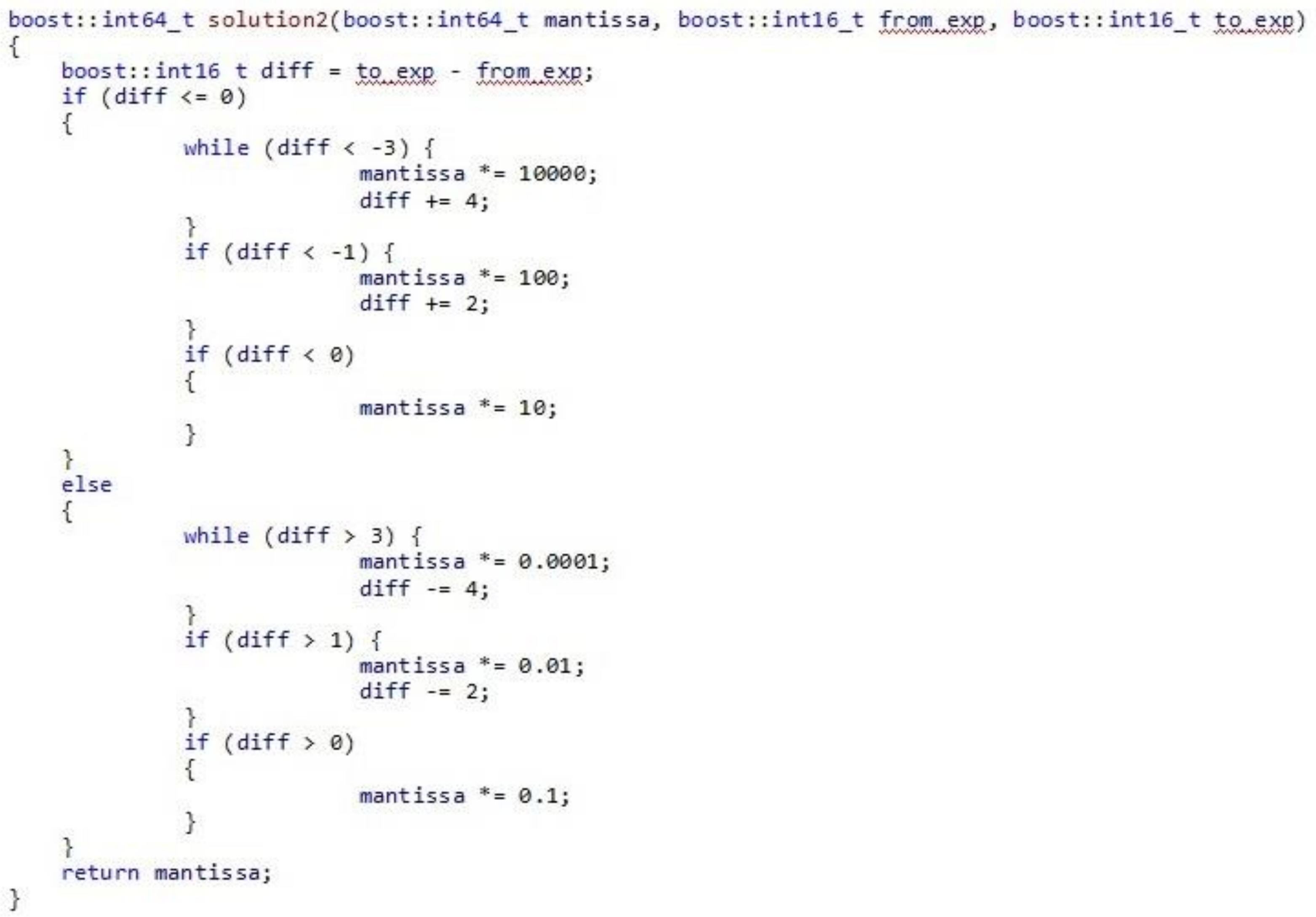
```
static boost::int64_t* MUL_POWERS_OF_10 = &MUL_POWERS_OF_10_BASE[9];
static boost::int64_t* DIV_POWERS_OF_10 = &DIV_POWERS_OF_10_BASE[9];

boost::int64_t solution1(boost::int64_t mantissa, boost::int16_t
  from_exp, boost::int16_t to_exp)
{
  boost::int16_t diff = from_exp - to_exp;
  return mantissa / DIV_POWERS_OF_10[diff] + mantissa *
        MUL_POWERS_OF_10[diff];
}
```

Solution 1(28ns):



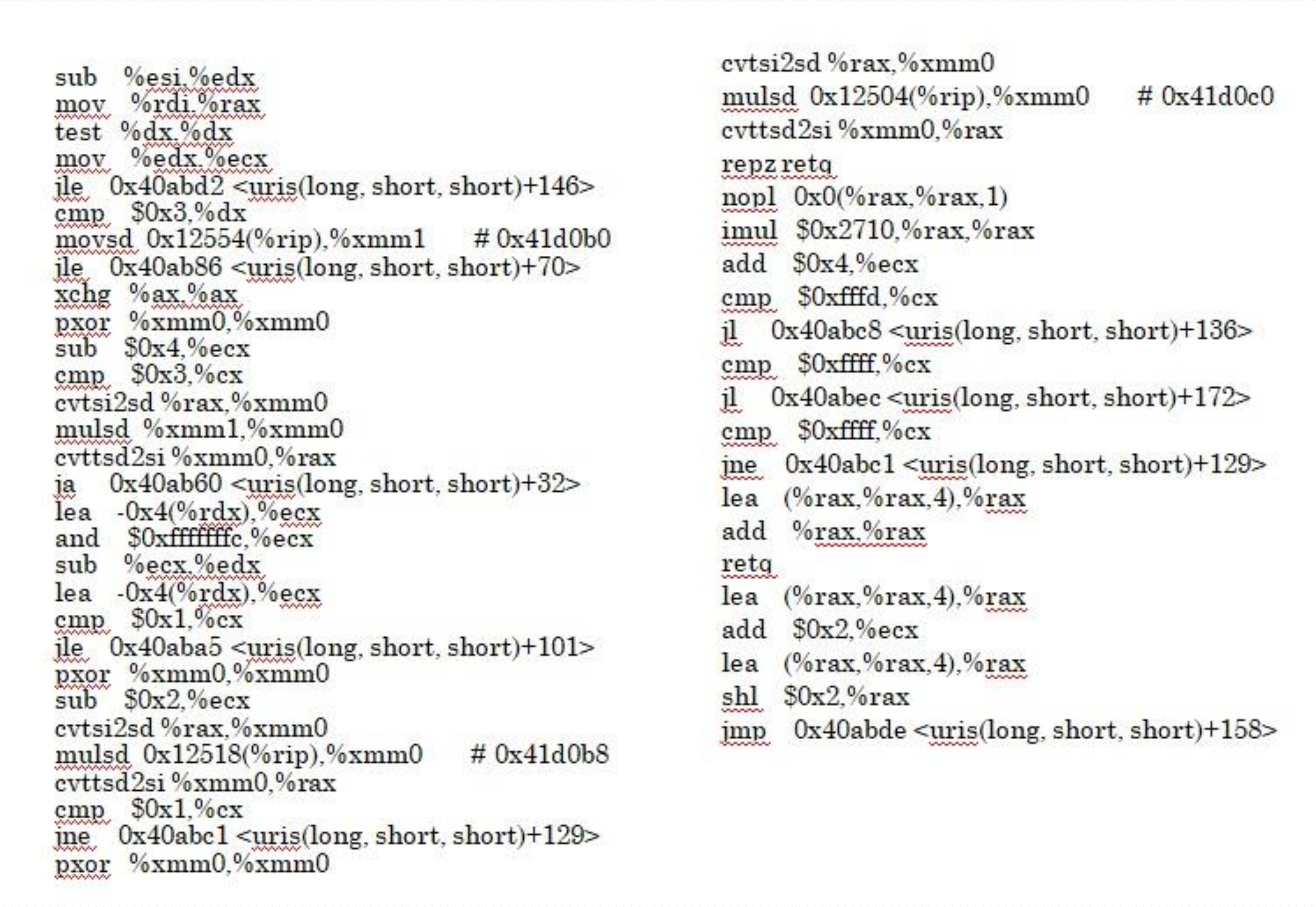
```
sub      %edx,%esi
mov      %rdi,%rax
movswq   %si,%rsi
cqto
imul     0x623428(,%rsi,8),%rdi
idivq    0x623388(,%rsi,8)
add      %rdi,%rax
retq
```

Solution 2:



```
boost::int64_t solution2(boost::int64_t mantissa, boost::int16_t from_exp, boost::int16_t to_exp)
{
    boost::int16_t diff = to_exp - from_exp;
    if (diff <= 0)
    {
            while (diff < -3) {
                    mantissa *= 10000;
                    diff += 4;
            }
            if (diff < -1) {
                    mantissa *= 100;
                    diff += 2;
            }
            if (diff < 0)
            {
                    mantissa *= 10;
            }
    }
    else
    {
            while (diff > 3) {
                    mantissa *= 0.0001;
                    diff -= 4;
            }
            if (diff > 1) {
                    mantissa *= 0.01;
                    diff -= 2;
            }
            if (diff > 0)
            {
                    mantissa *= 0.1;
            }
    }
    return mantissa;
}
```

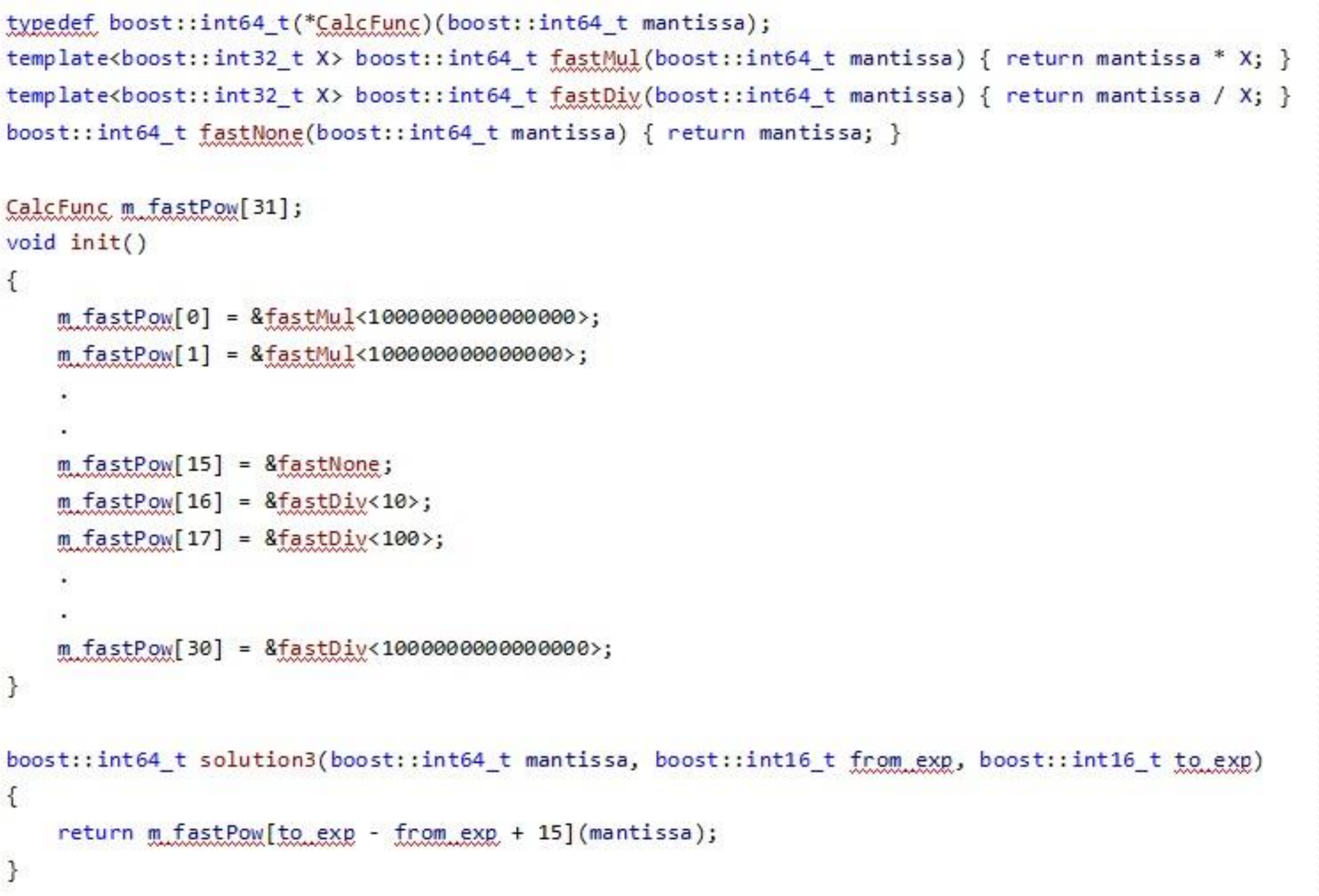
Solution 2 (23ns):



```
sub   %esi,%edx
mov   %rdi,%rax
test  %dx,%dx
mov   %edx,%ecx
jle   0x40abd2 <uris(long, short, short)+146>
cmp   $0x3,%dx
movsd 0x12554(%rip),%xmm1      # 0x41d0b0
jle   0x40ab86 <uris(long, short, short)+70>
xchg  %ax,%ax
pxor  %xmm0,%xmm0
sub   $0x4,%ecx
cmp   $0x3,%cx
cvtsi2sd %rax,%xmm0
mulsd %xmm1,%xmm0
cvttsd2si %xmm0,%rax
ja    0x40ab60 <uris(long, short, short)+32>
lea   -0x4(%rdx),%ecx
and   $0xfffffffc,%ecx
sub   %ecx,%edx
lea   -0x4(%rdx),%ecx
cmp   $0x1,%cx
jle   0x40aba5 <uris(long, short, short)+101>
pxor  %xmm0,%xmm0
sub   $0x2,%ecx
cvtsi2sd %rax,%xmm0
mulsd 0x12518(%rip),%xmm0      # 0x41d0b8
cvttsd2si %xmm0,%rax
cmp   $0x1,%cx
jne   0x40abc1 <uris(long, short, short)+129>
pxor  %xmm0,%xmm0
```

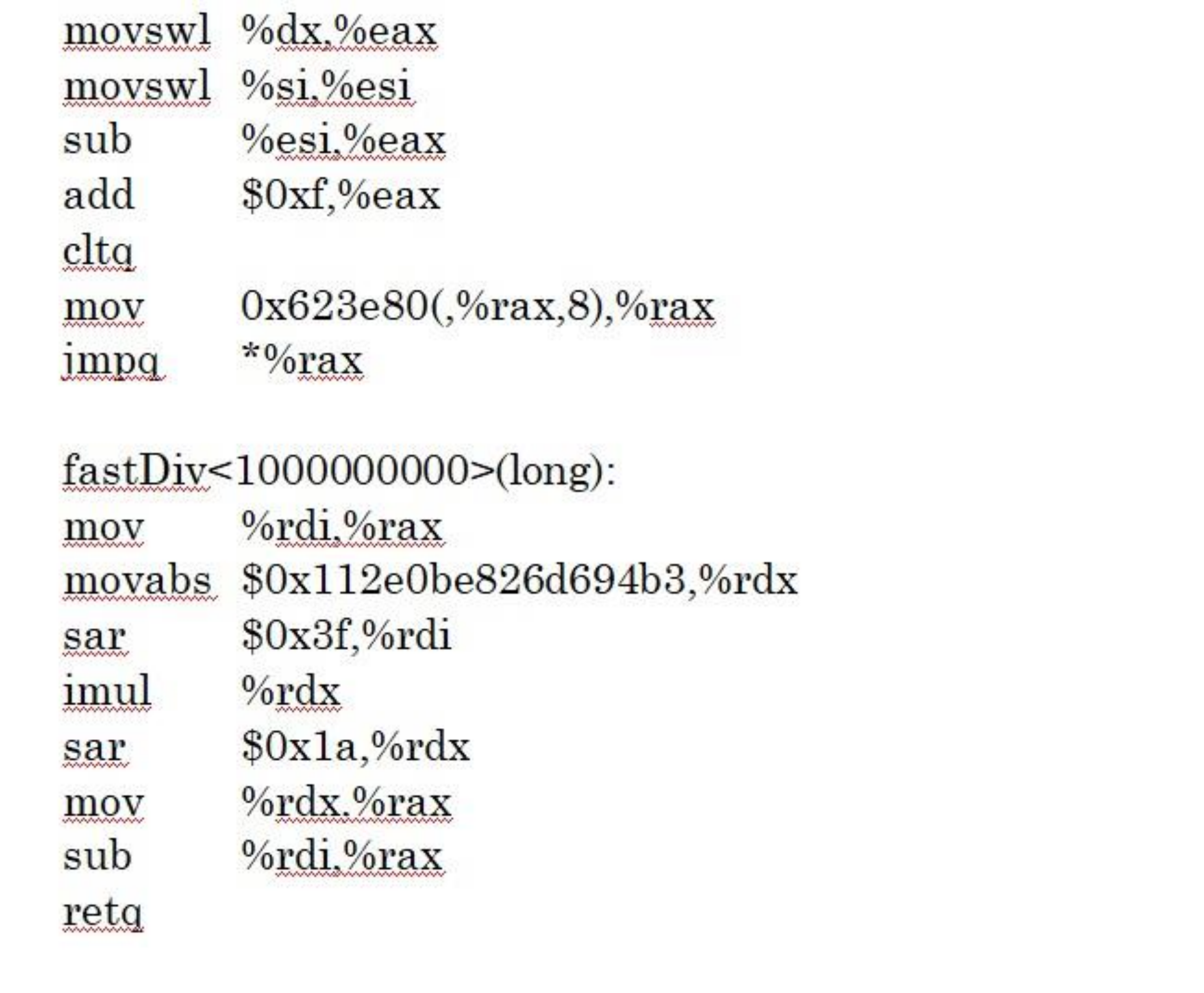
```
cvtsi2sd %rax,%xmm0
mulsd 0x12504(%rip),%xmm0      # 0x41d0c0
cvttsd2si %xmm0,%rax
repz retq
nopl  0x0(%rax,%rax,1)
imul  $0x2710,%rax,%rax
add   $0x4,%ecx
cmp   $0xfffd,%cx
jl    0x40abc8 <uris(long, short, short)+136>
cmp   $0xffff,%cx
jl    0x40abec <uris(long, short, short)+172>
cmp   $0xffff,%cx
jne   0x40abc1 <uris(long, short, short)+129>
lea   (%rax,%rax,4),%rax
add   %rax,%rax
retq
lea   (%rax,%rax,4),%rax
add   $0x2,%ecx
lea   (%rax,%rax,4),%rax
shl   $0x2,%rax
jmp   0x40abde <uris(long, short, short)+158>
```

Solution 3:



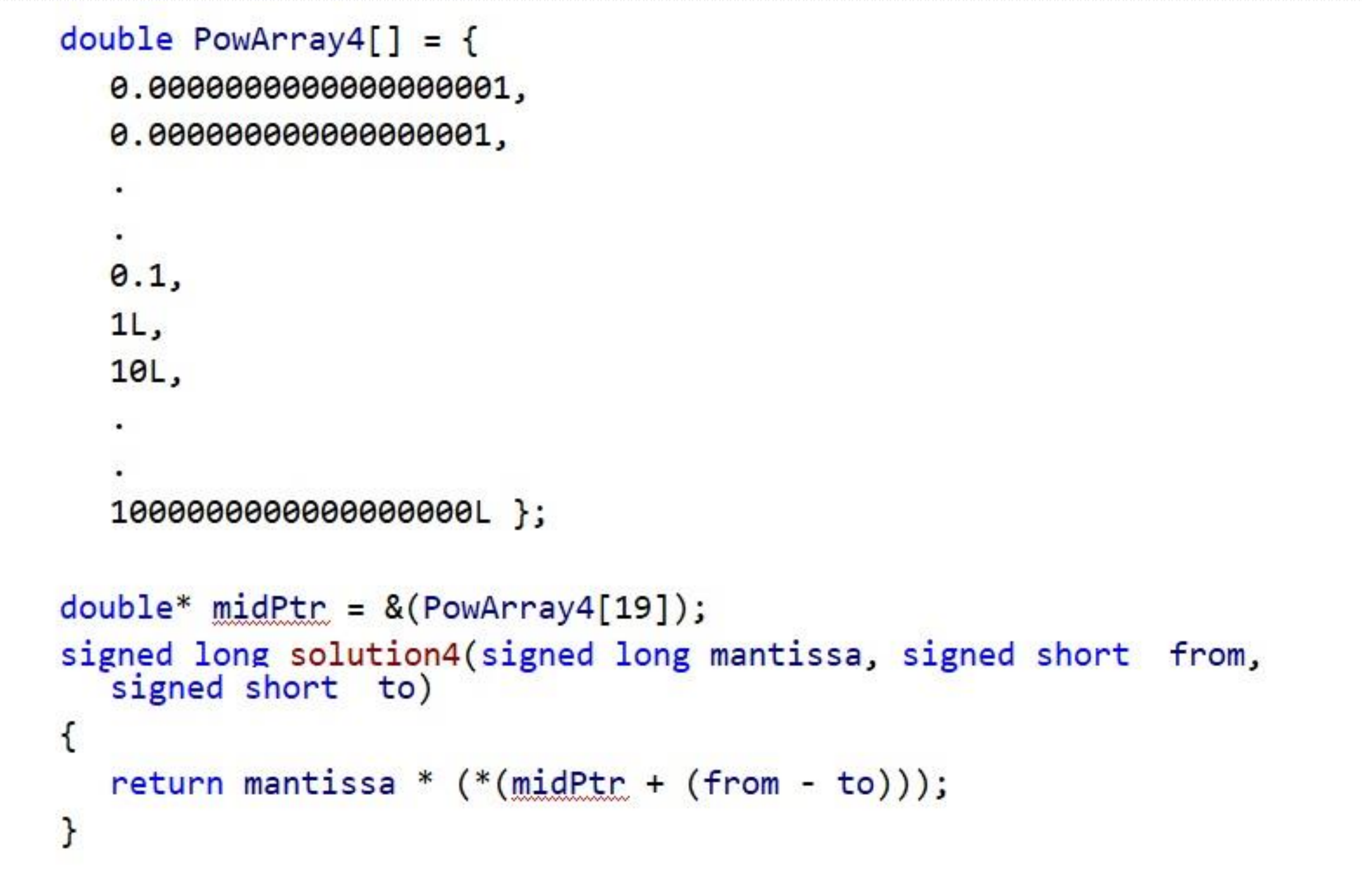
```
typedef boost::int64_t(*CalcFunc)(boost::int64_t mantissa);
template<boost::int32_t X> boost::int64_t fastMul(boost::int64_t mantissa) { return mantissa * X; }
template<boost::int32_t X> boost::int64_t fastDiv(boost::int64_t mantissa) { return mantissa / X; }
boost::int64_t fastNone(boost::int64_t mantissa) { return mantissa; }

CalcFunc m_fastPow[31];
void init()
{
    m_fastPow[0] = &fastMul<1000000000000000>;
    m_fastPow[1] = &fastMul<100000000000000>;
    .
    .
    m_fastPow[15] = &fastNone;
    m_fastPow[16] = &fastDiv<10>;
    m_fastPow[17] = &fastDiv<100>;
    .
    .
    m_fastPow[30] = &fastDiv<1000000000000000>;
}

boost::int64_t solution3(boost::int64_t mantissa, boost::int16_t from_exp, boost::int16_t to_exp)
{
    return m_fastPow[to_exp - from_exp + 15](mantissa);
}
```

Solution 3 (20ns):



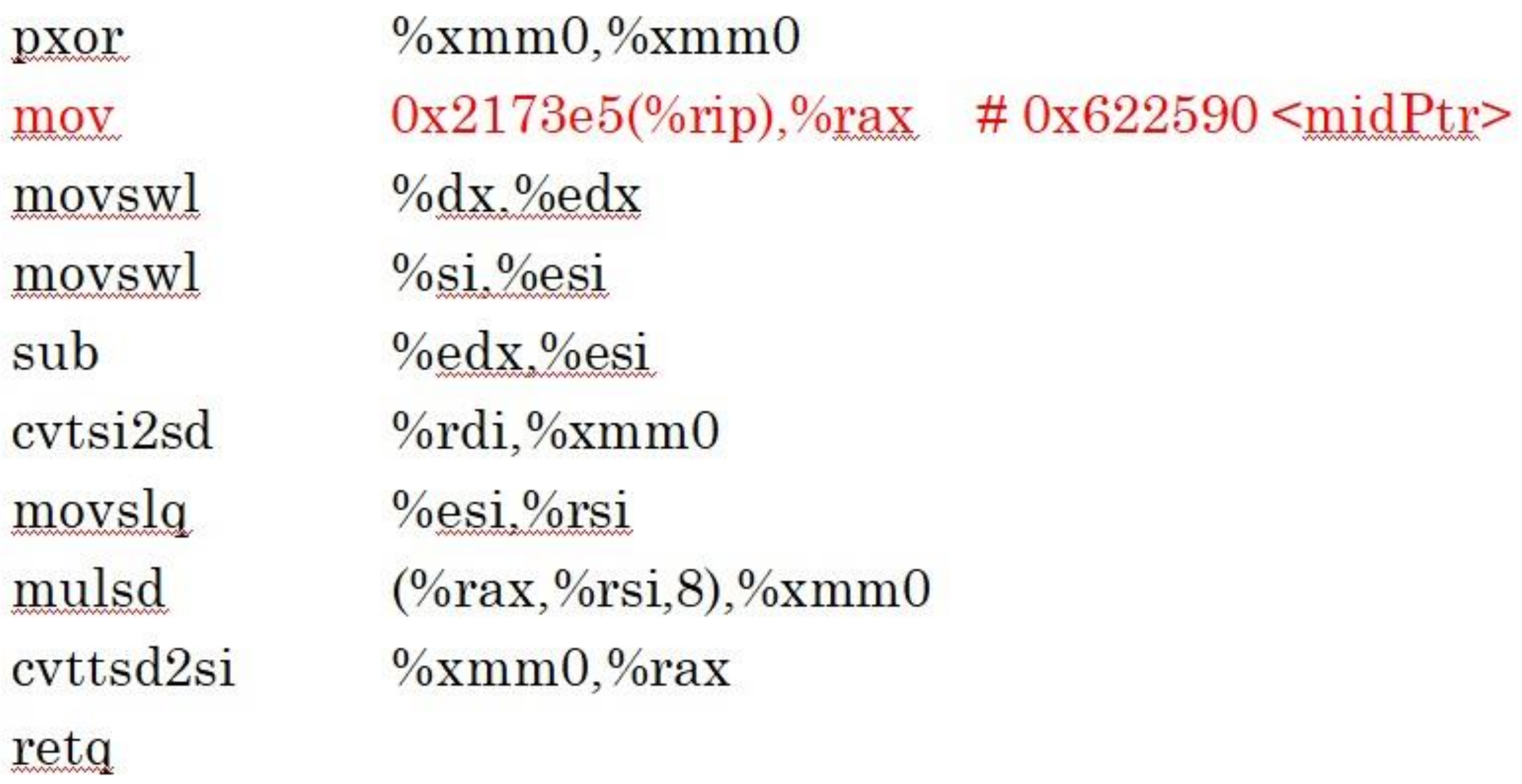
```
movswl  %dx,%eax
movswl  %si,%esi
sub     %esi,%eax
add     $0xf,%eax
cltq
mov     0x623e80(,%rax,8),%rax
jmpq    *%rax

fastDiv<1000000000>(long):
mov     %rdi,%rax
movabs  $0x112e0be826d694b3,%rdx
sar     $0x3f,%rdi
imul    %rdx
sar     $0x1a,%rdx
mov     %rdx,%rax
sub     %rdi,%rax
retq
```

Solution 4:



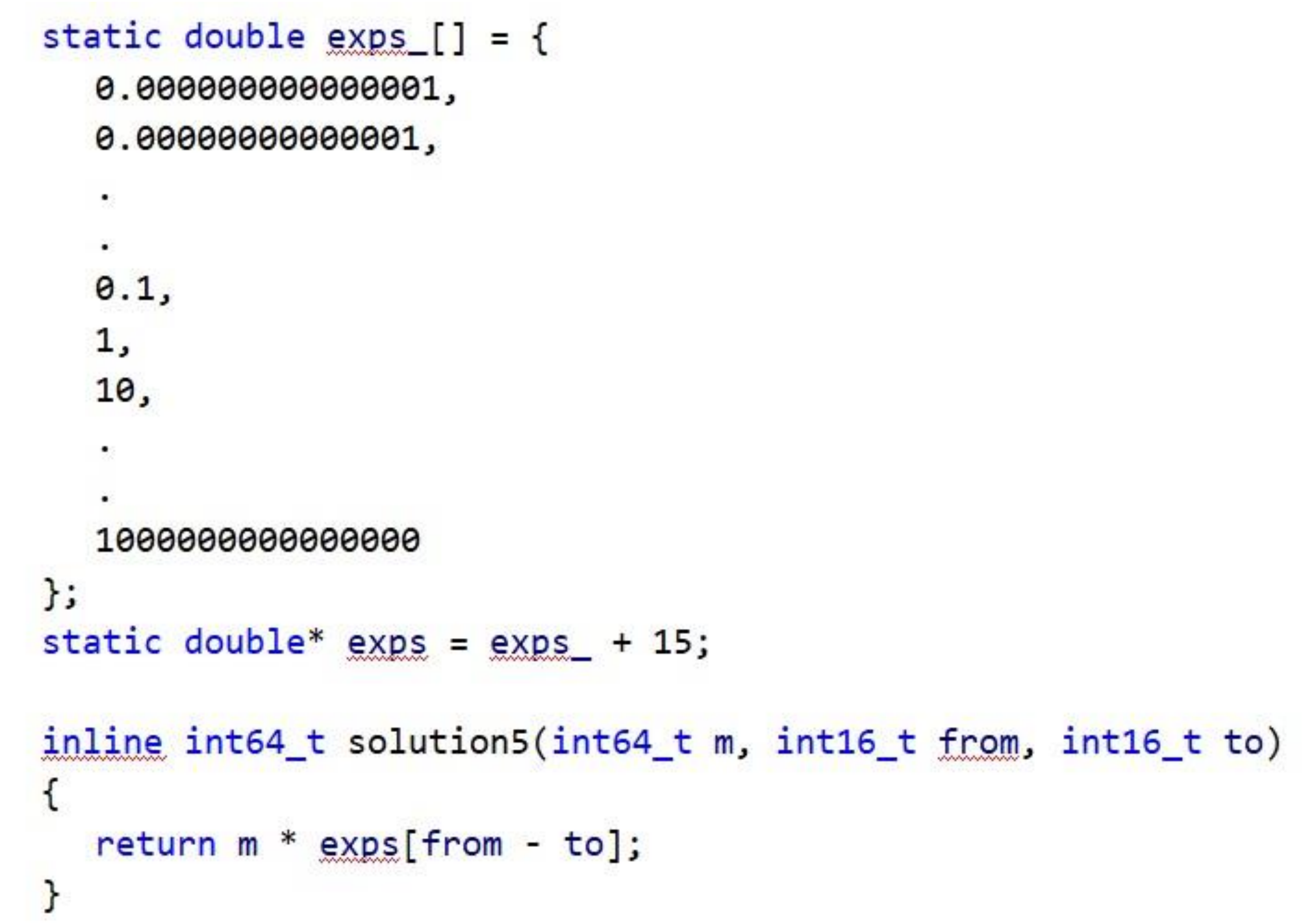
```
double PowArray4[] = {
   0.0000000000000000001,
   0.000000000000000001,
   .
   .
   0.1,
   1L,
   10L,
   .
   .
   1000000000000000000L };

double* midPtr = &(PowArray4[19]);
signed long solution4(signed long mantissa, signed short  from,
   signed short  to)
{
   return mantissa * (*(midPtr + (from - to)));
}
```

Solution 4 (16.8ns):



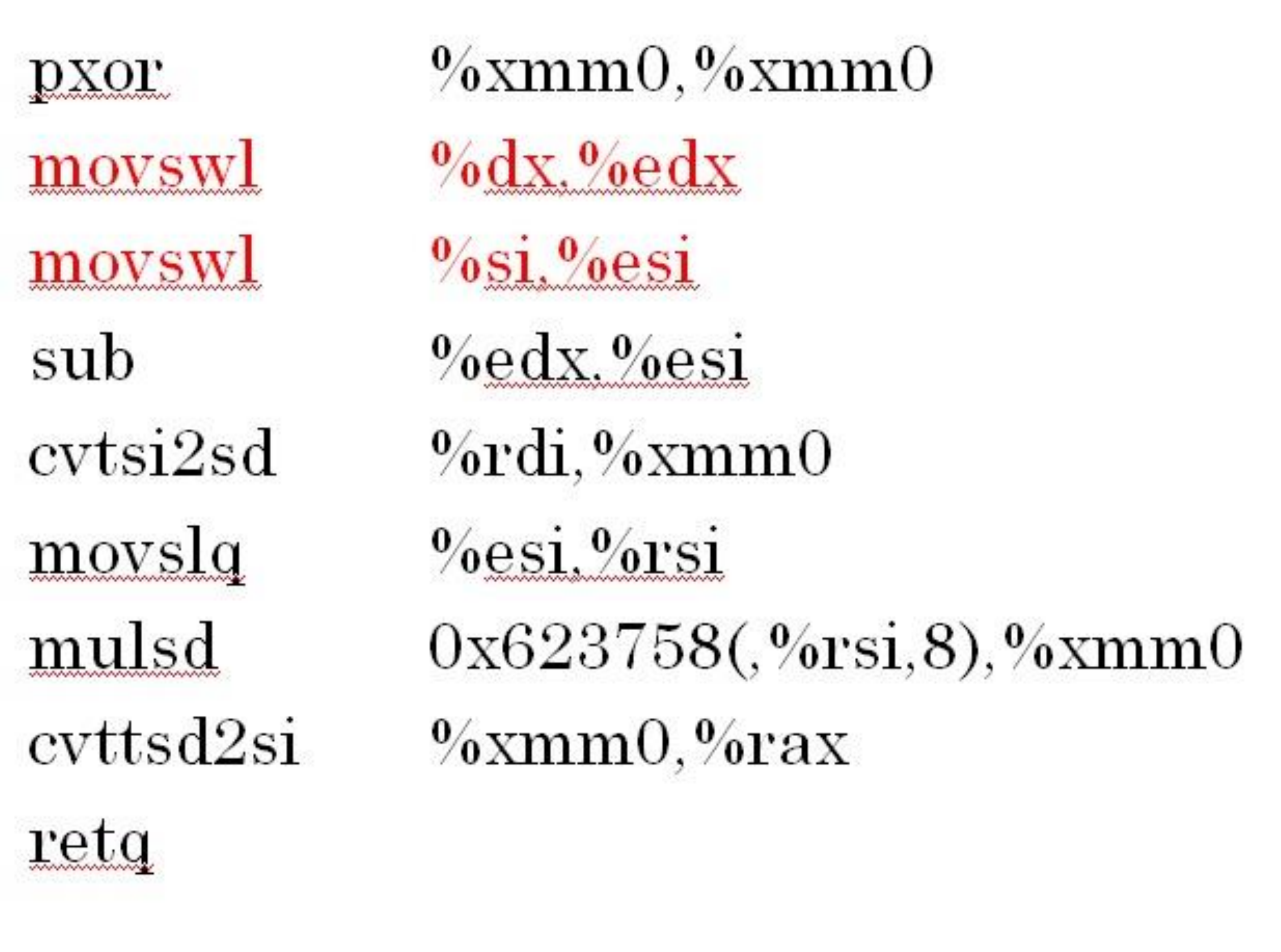
```
pxor        %xmm0,%xmm0
mov         0x2173e5(%rip),%rax   # 0x622590 <midPtr>
movswl      %dx,%edx
movswl      %si,%esi
sub         %edx,%esi
cvtsi2sd    %rdi,%xmm0
movslq      %esi,%rsi
mulsd       (%rax,%rsi,8),%xmm0
cvttsd2si   %xmm0,%rax
retq
```

Solution 5:



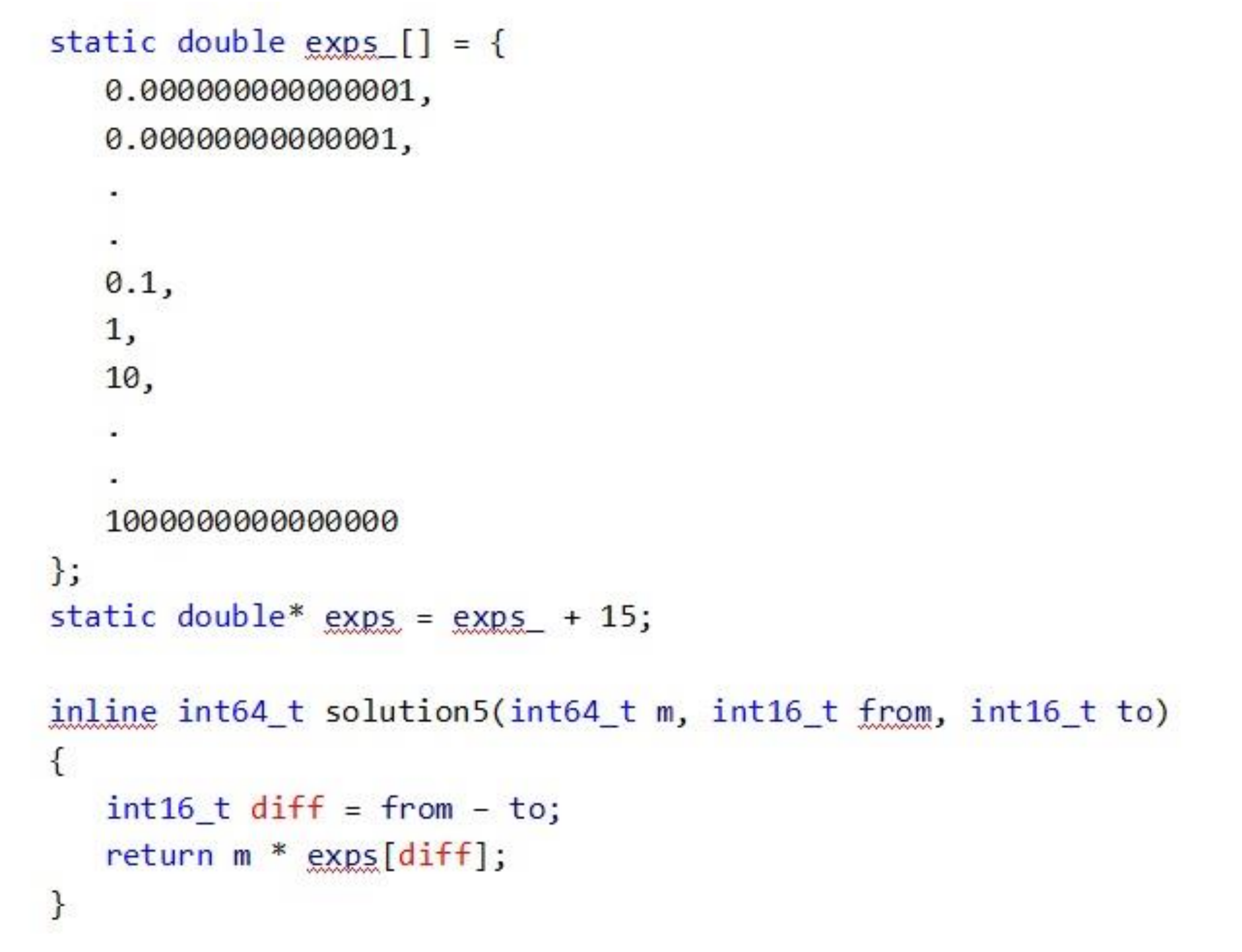
```
static double exps_[] = {
   0.000000000000001,
   0.00000000000001,
   .
   .
   0.1,
   1,
   10,
   .
   .
   1000000000000000
};
static double* exps = exps_ + 15;

inline int64_t solution5(int64_t m, int16_t from, int16_t to)
{
   return m * exps[from - to];
}
```

Solution 5 (16.4ns):



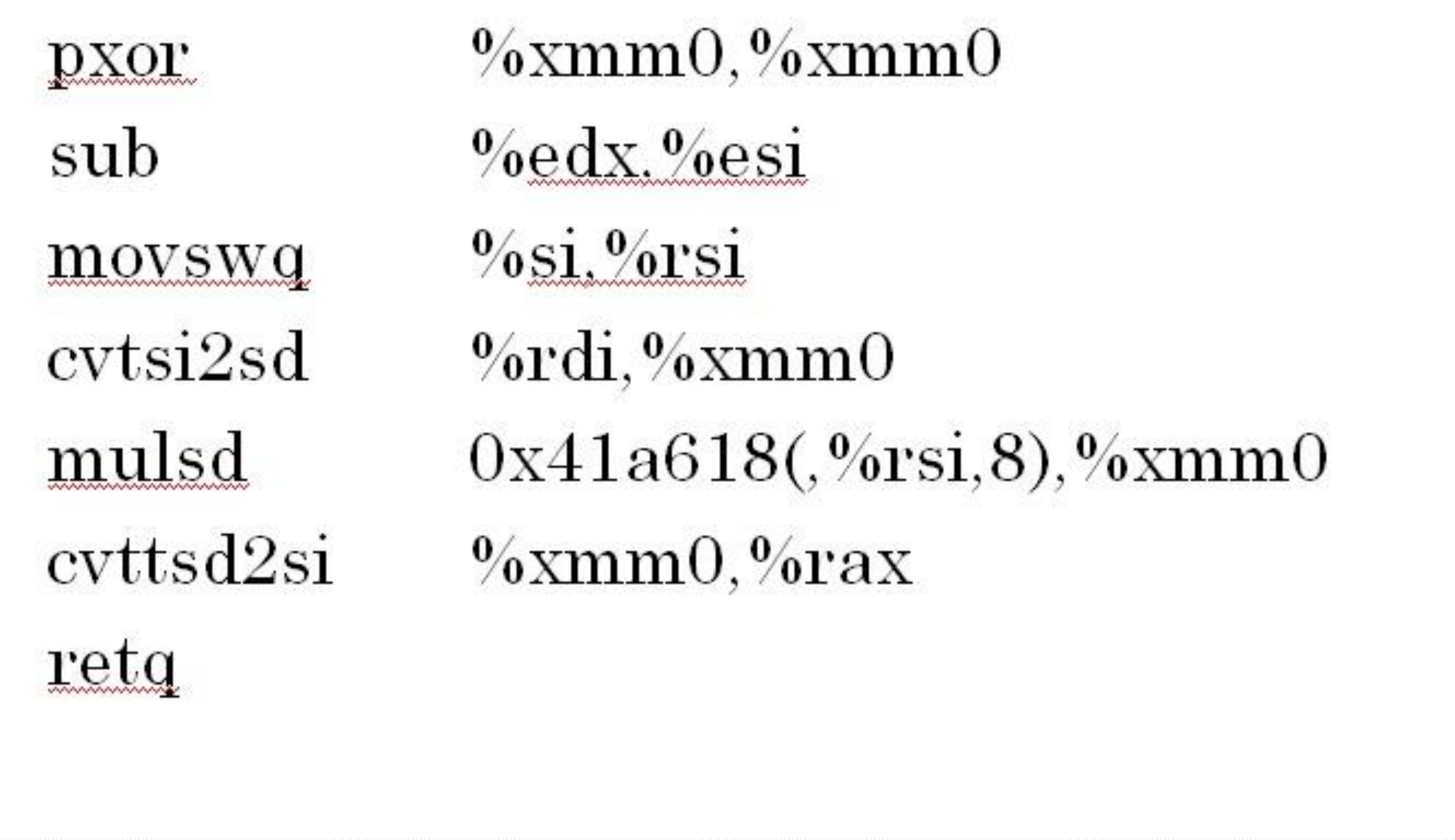
```
pxor        %xmm0,%xmm0
movswl      %dx.%edx
movswl      %si.%esi
sub         %edx.%esi
cvtsi2sd    %rdi,%xmm0
movslq      %esi.%rsi
mulsd       0x623758(,%rsi,8),%xmm0
cvttsd2si   %xmm0,%rax
retq
```

Solution 6:



```
static double exps_[] = {
   0.000000000000001,
   0.00000000000001,
   .
   .
   0.1,
   1,
   10,
   .
   .
   1000000000000000
};
static double* exps = exps_ + 15;

inline int64_t solution5(int64_t m, int16_t from, int16_t to)
{
   int16_t diff = from - to;
   return m * exps[diff];
}
```

Solution 6 (16 ns):



```
pxor        %xmm0,%xmm0
sub         %edx.%esi
movswq      %si,%rsi
cvtsi2sd    %rdi,%xmm0
mulsd       0x41a618(,%rsi,8),%xmm0
cvttsd2si   %xmm0,%rax
retq
```

Solution 7:

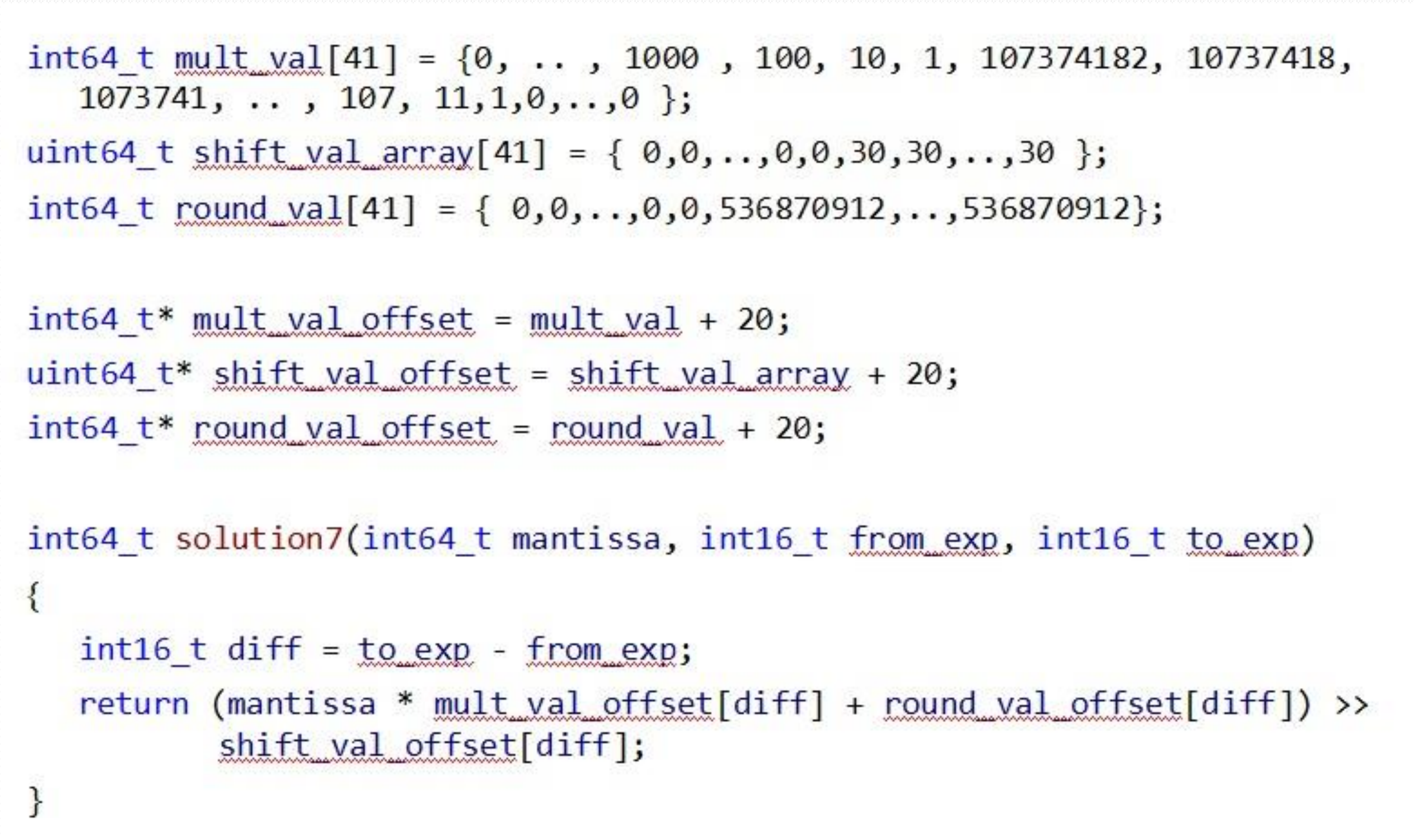
$$\frac{X}{1000} = \left(X \cdot \frac{2^{32}}{1000} \right) \div 2^{32} = \left(X \cdot 4294967.296 \right) \div 2^{32}$$

$$\approx \left(X \cdot 4294967 \right) \div 2^{32} = \left(X \cdot 4294967 \right) >> 32$$

General case: $$\frac{X}{C} = \left(X \cdot \left\lfloor \frac{2^k}{C} \right\rfloor \right) >> K$$

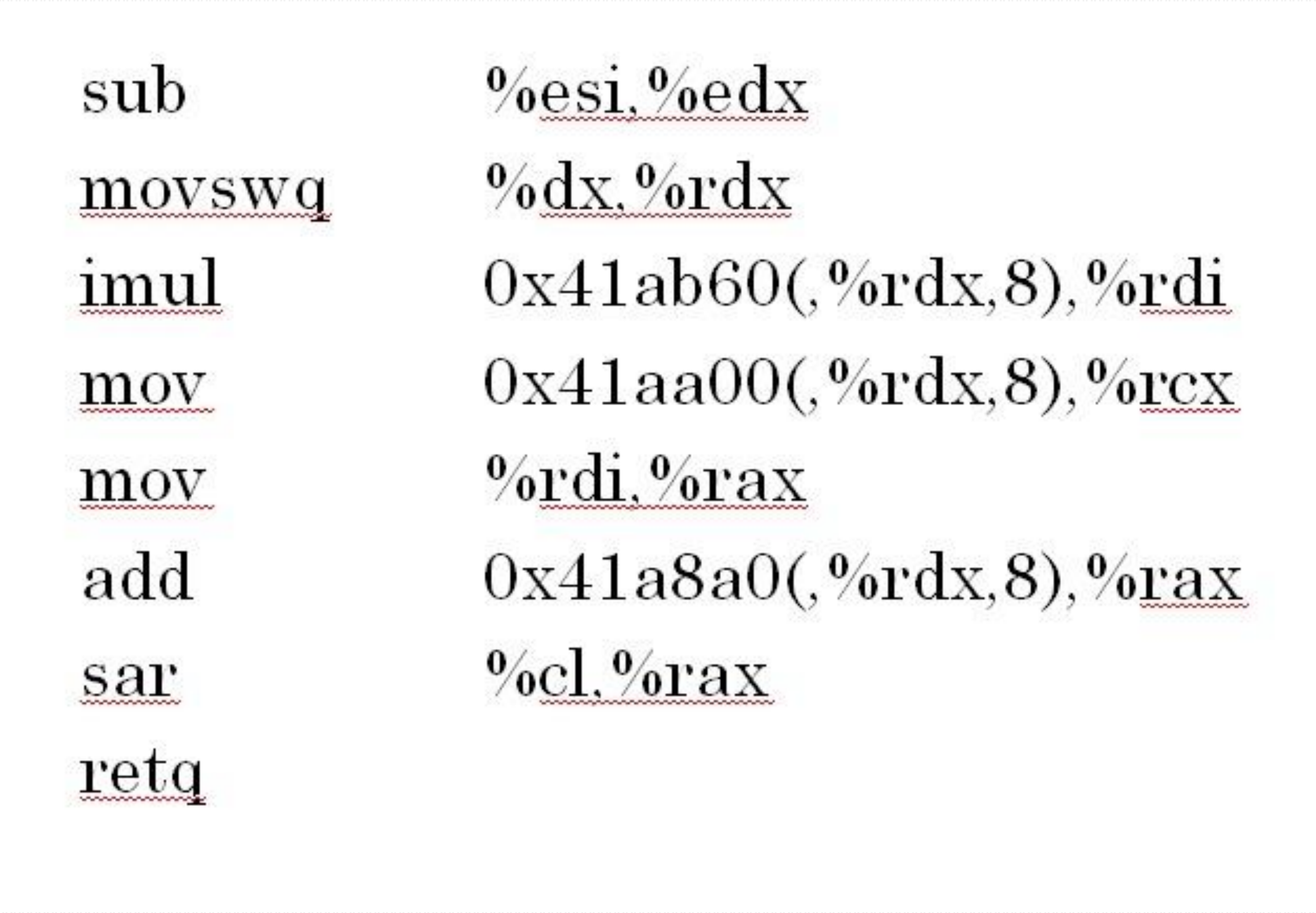
Error: $\dfrac{X}{2^k}$

Solution 7:



```
int64_t mult_val[41] = {0, .. , 1000 , 100, 10, 1, 107374182, 10737418,
   1073741, .. , 107, 11,1,0,..,0 };
uint64_t shift_val_array[41] = { 0,0,..,0,0,30,30,..,30 };
int64_t round_val[41] = { 0,0,..,0,0,536870912,..,536870912};

int64_t* mult_val_offset = mult_val + 20;
uint64_t* shift_val_offset = shift_val_array + 20;
int64_t* round_val_offset = round_val + 20;

int64_t solution7(int64_t mantissa, int16_t from_exp, int16_t to_exp)
{
   int16_t diff = to_exp - from_exp;
   return (mantissa * mult_val_offset[diff] + round_val_offset[diff]) >>
           shift_val_offset[diff];
}
```

Solution 7 (15 ns):



```
sub       %esi,%edx
movswq    %dx,%rdx
imul      0x41ab60(,%rdx,8),%rdi
mov       0x41aa00(,%rdx,8),%rcx
mov       %rdi,%rax
add       0x41a8a0(,%rdx,8),%rax
sar       %cl,%rax
retq
```

- HFT ecosystem has drastically changed over the last decades
- Trading system are required to work with low latency and low jitter
- There are several factors which might impact our system latency and jitter aside from the code
- Measurement is a must, sometimes thing are counter intuitive
- Looking at the assembly which is generated by the compiler helps
- Call for a challenge (www.final.co.il, contactus@final.co.il)

Thank
you!!