**ETH login ID:**
(Please print in capital letters)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |

**Full name:** _____

**263-2300: How to Write Fast Numerical Code**
ETH Computer Science, Spring 2017
Midterm Exam
Wednesday, April 26, 2017

**Instructions**

- Make sure that your exam is not missing any sheets, then write your full name and login ID on the front.
- No extra sheets are allowed.
- The exam has a maximum score of 100 points.
- No books, notes, calculators, laptops, cell phones, or other electronic devices are allowed.

| | |
|---|---|
| Problem 1 ($12 = 2 + 5 + 5$) | |
| Problem 2 ($13 = 3 + 10$) | |
| Problem 3 ($20 = 4 + 8 + 4 + 4$) | |
| Problem 4 ($18 = 6 + 6 + 6$) | |
| Problem 5 ($20 = 4 + 3 + 3 + 4 + 3 + 3$) | |
| Problem 6 ($17 = 2 + 2 + 3 + 3 + 2 + 3 + 2$) | |
| **Total** (100) | |

# Problem 1: Operational Intensity (12 = 2 + 5 + 5)

Consider a function that multiplies $n \times n$ matrices of doubles (`sizeof(double) = 8`)) in the form $C = AB + C$ implemented as straightforward triple loop. The function is run on a computer with a last level cache (write-back/write-allocate) of size 256 KB. Assume an initially empty cache for the below questions.

1. What is the flop count $W(n)$?

2. Determine a lower bound for the data movement $Q(n)$ (in bytes) incurred by the function considering compulsory reads and writes. Then use the result to obtain an upper bound for $I(n) = W(n)/Q(n)$. Show enough detail so we can see your reasoning.

3. For which sizes $n$ would you expect the previous upper bound to be (roughly) the accurate value for $I(n)$? Show your derivation and reasoning.

# Problem 2: Flop Count (13 = 3 + 10)

Consider the following code for computing the determinant of an invertible matrix using Bareiss algorithm. Assume that `N` $> 1$.

```
1  void bareiss(float **A, int N){
2      int i, j, k;
3      for (i = 0; i < N-1; i += 1) {
4          for (j = i + 1; j < N; j += 1){
5              for (k = i + 1; k < N; k += 1) {
6                  A[j][k] = A[j][k] * A[i][i] - A[j][i] * A[i][k];
7                  if (i == 2) A[j][k] /= A[i-1][i-1];
8              }
9          }
10     }
11 }
```

1. Define a detailed floating point cost measure $C(N)$ for the function `bareiss`. Ignore integer operations.

2. Compute the cost $C(N)$ as just defined. Show your derivation.

   **Note:** Lower-order terms (and only those) may be expressed using big-O notation. This means: as the final result something like $3n + O(\log(n))$ would be ok but $O(n)$ is not.

   The following formulas may be helpful:

   - $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2}{2} + O(n)$
   - $\sum_{i=0}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6} = \frac{n^3}{3} + O(n^2)$

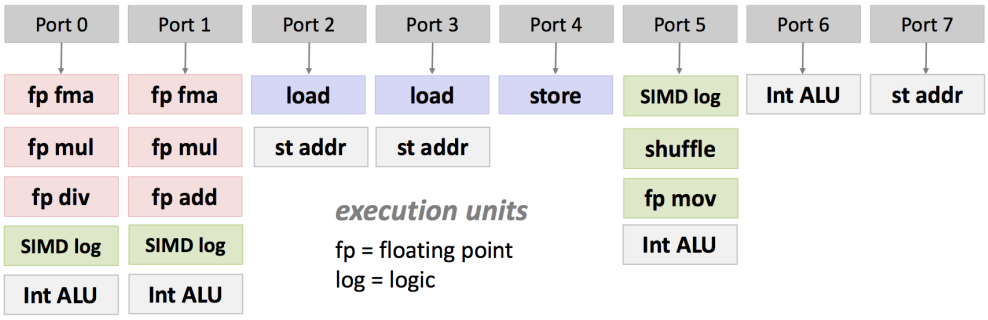
# Problem 3: Bounds (20 = 4 + 8 + 4 + 4)

Consider the AVX code below that performs point-wise multiplication of two arrays of interleaved complex numbers:

```
1  #include <immintrin.h>
2
3  //
4  // Assume 'neg' is a global variable
5  //
6  __m256d neg;
7
8  //
9  // Assume that 'init_f' will be called once before 'f'
10 //
11 void init_f () {
12     const double global_neg [] = {1.0, -1.0, 1.0, -1.0};
13     neg = _mm256_loadu_pd(global_neg);
14 }
15
16 //
17 // Assume that N is divisible by 2
18 //
19 void f (const double * lhs, const double * rhs, double * res, size_t N)
20 {
21     __m256d va, vb, v1, v2, v3, v4, v5;
22     size_t i;
23
24     for (i = 0; i < 2 * N; i += 4)
25     {
26         va = _mm256_loadu_pd   (lhs + i);
27         vb = _mm256_loadu_pd   (rhs + i);
28
29         v1 = _mm256_mul_pd     (va, vb);
30         v2 = _mm256_permute_pd (vb, 0x5);
31         v3 = _mm256_mul_pd     (v2, neg);
32         v4 = _mm256_mul_pd     (va, v3);
33         v5 = _mm256_hsub_pd    (v1, v4);
34
35         _mm256_storeu_pd       (res + i, v5);
36     }
37 }
```

Assume that the code is executed on Haswell computer, such that the whole working set of function `f` fits L1 cache and is already loaded into L1 (warm cache scenario). Also assume that N is always divisible by 2, and assume that the `init_f` function has been executed once in the main function to initialize the global variable `neg`.

Figure 1 shows the port structure of the Haswell microarchitecture, and information on SIMD intrinsics used in the code above. Integer operations can be ignored in this question. **Show enough detail with each answer so we understand your reasoning.**




| Instruction | Latency [cycles] | Max. throughput (all ports) [instructions/cycle] | Port |
|---|---|---|---|
| `_mm256_storeu_pd` | 1 | 1 | 4 |
| `_mm256_loadu_pd` | 1 | 2 | 2/3 |
| `_mm256_permute_pd` | 1 | 1 | 5 |
| `_mm256_mul_pd` | 5 | 2 | 0/1 |
| `_mm256_hsub_pd` | 5 | 1/2 | 1 |

Figure 1: Dispatch Port and Execution Stacks of the Haswell Microarchitecture and performance profile of several SIMD instructions

1. Determine the (mathematical) cost of `f` measured in flops.

2. Determine a lower bound (as tight as possible) for the runtime and an associated upper bound for the performance of `f` based on the instruction mix, ignoring dependencies between instructions (i.e., don't consider latencies and assume full throughput).

3. Take now into consideration the dependencies: draw a DAG for one loop iteration (i.e., for the lines 26–35). The nodes are the instructions.

4. Based on 3. above estimate the runtime (latency) of one loop iteration using latency information.

# Problem 4: Cache Mechanics (18 = 6 + 6 + 6)

We consider a 128 byte data cache that is fully associative and can hold 4 doubles in every cache line. The cache uses least recently used replacement policy. A double is assumed to require 8 bytes.

For the `C` code below we assume a cold cache. The code accesses elements of array `A` in the order determined by function `f`. Assume that `A` is randomly initialized, cache aligned (that is, `A[0]` is loaded into the first slot of a cache line) and that all scalar variables are held in registers.

**Note:** It helps to draw the cache.

```
1 int i, j;
2 double A[55];
3 for (i = 0; i < 55; i++)
4     A[f(i)] = A[f(i)] + i;
```

Consider the reads and the writes and answer the following:

1. For `f(i) = i`:
   (a) Determine the miss rate.
   (b) What kind of misses occur?
   (c) What kind of locality does the code have with respect to accesses of `A` and this cache.
2. For `f(i) = (2*i)%55`:
   (a) Determine the miss rate.
   (b) What kind of misses occur?

(c) What kind of locality does the code have with respect to accesses of `A` and this cache.

3. For `f(i) = (21*i)%55`:

   (a) Determine the miss rate.

   (b) What kind of misses occur?

   (c) What kind of locality does the code have with respect to accesses of `A` and this cache?

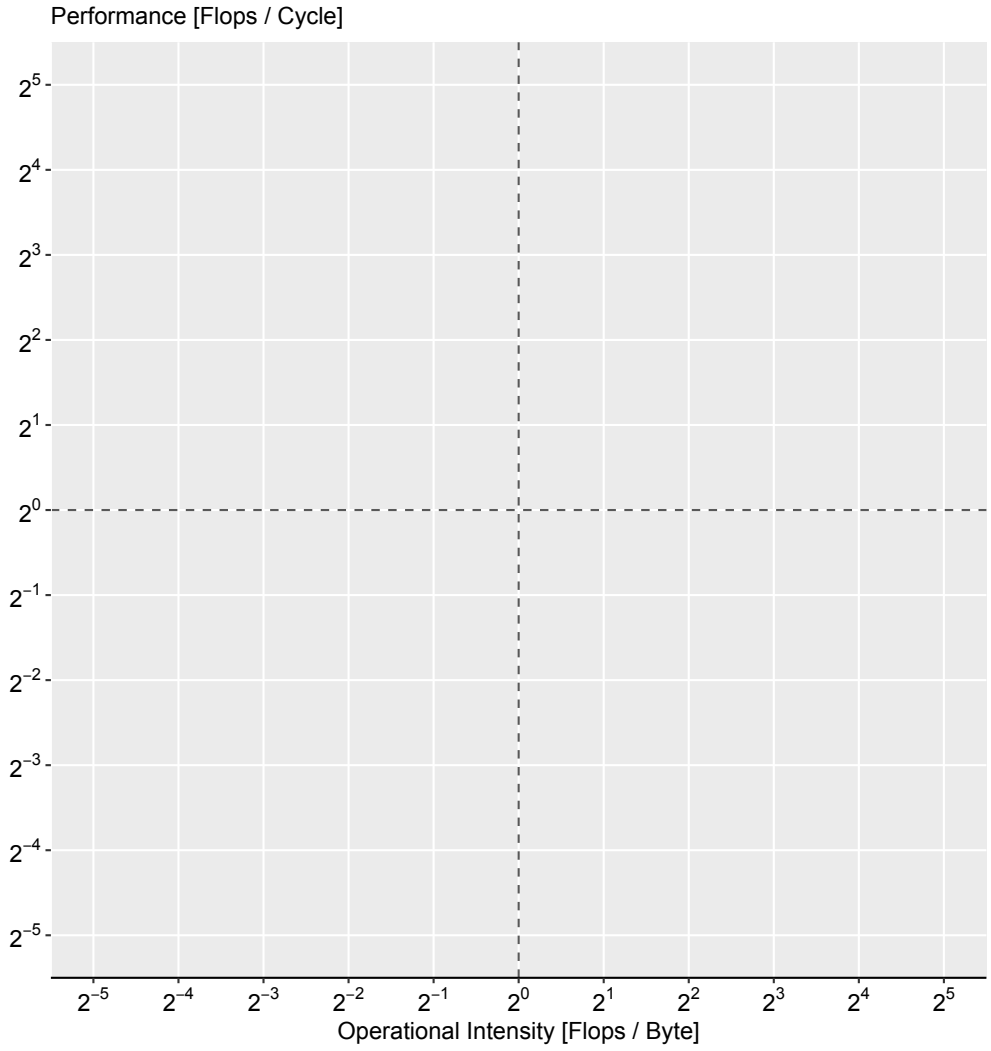
# Problem 5: Roofline (20 = 4 + 3 + 3 + 4 + 3 + 3)

Given is a computer that supports single precision floating point operations (`sizeof(float) = 4`). It does not support any SIMD operations. The relevant port information is as follows:

Port 0: fma, mul

Port 1: fma, mul, add

Each of these operations has a throughput of 1 per port. The memory bandwidth is 4 floats per cycle. The cache block size is 4 floats and the cache is write-back/write-allocate. We assume a cold cache scenario.




1. Draw the roofline plot for this computer into the above graph. Give a very brief explanation.

Now we consider the following function, where `k` is a global variable. `x` and `y` are not aliased.

```
1 // assume k is defined globally
2 void f(float * x, float * y, int n) {
3   int i; float a, b;
4   for (i = 0; i < k * n; i += k) {
5     a = x[i];
6     b = y[i];
7     y[i] = a * (2 * a + b);
8   }
9 }
```

2. Based (only) on the instruction mix (i.e. ignoring dependencies), which performance is maximally achievable for this function and why? Draw an associated tighter horizontal roof into the plot above.

3. At what operational intensity $I$ does this new horizontal roof intersect with the memory roof?

4. Assume `k = 1`. Determine the operational intensity (reads and writes) $I$ of `f` and mark it in the plot. Based on this $I$, which peak performance is achievable?

5. Assume now `k = 2`. Determine the operational intensity (reads and writes) $I$ of `f` and mark it in the plot. Based on this $I$, which peak performance is achievable?

6. Give a general formula for $I$, dependent on `k`.

# Problem 6: Sampler (17 = 2 + 2 + 3 + 3 + 2 + 3 + 2)

1. Provide row_start($M$) of the below matrix when expressed in Compressed Sparse Row (CSR) format.

$$M = \begin{pmatrix} 0 & 1 & 3 & 7 \\ 0 & 0 & 5 & 0 \\ 0 & 2 & 6 & 0 \end{pmatrix}$$

2. Name two processor mechanisms that can dynamically increase the ILP of an executable.

3. A function computing $y = 2 * x + y$, where $x, y$ are vectors of doubles of length $n$ runs on a processor that can execute per cycle one floating point add and one floating point mult. Running with cold cache once, the function achieves 25% of peak. Estimate the memory read bandwidth.

4. Assume two functions $f_1$, $f_2$ that both implement matrix multiplication. $f_1$ is implemented (and run) with optimal square blocking on a computer with a last level cache of size $\gamma_1$. $f_2$ is implemented (and run) with optimal square blocking on a computer with a last level cache of size $\gamma_2$. Estimate the ratio of the operational intensities $I_1, I_2$ of $f_1, f_2$ for large $n$.

5. Why does a numerical function typically need more ILP (independent operations) than available execution units to achieve high performance on modern processors (e.g., on Intel Haswell)?

6. Given is a computer with one cache of size 512 bytes and a cache block size of 16 bytes. How many entries should a TLB have at least so that cache hits imply TLB hits? (Ignore conflicts due to associativity.)

7. Name two types of dependencies that are avoided if C code is written in SSA style?