University of Western Ontario, Computer Science Department
CS3350B, Computer Organization

Assignment 1 **Due: Friday, February 02, 2023**

General Instructions: This assignment consists of 14 pages, 5 exercises, and is marked out of 100. For any question involving calculations you must provide your workings. *Correct final answers without workings will be marked as incorrect.* Each assignment submission and the answers within are to be solely your individual work and completed independently. Any plagiarism found will be taken seriously and may result in a mark of 0 on this assignment, removal from the course, or more serious consequences.

Submission Instructions: The answers to this assignment are to be submitted electronically to Gradescope. Ideally, the answers are to be typed. At the very least, clearly *scanned* copies (no photographs) of hand-written work. If the person correcting your assignment is unable to easily read or interpret your answer then it may be marked as incorrect without the possibility of remarking.

`https://www.gradescope.ca/courses/9882`

Useful Facts:

$$1\ GHz = 1 \times 10^9\ Hz$$

$$1\ byte = 8\ bits$$

$$1\ Kbyte\ (KB) = 1024\ bytes$$

Exercise 1. **[15 Marks]** Consider the following two functions: `factRec` and `factIter`, which compute the factorial of a number.

Discuss the *instruction* locality of each the function. Is there good locality or bad locality? Is there spatial locality or temporal locality? **Explain why.** For ease of discussion, you may consider a particular execution of the function, say, `n = 5`.

```
int factRec(int n) {
    if (n < 1) {
        return 1;
    }
    int n1 = factRec(n-1);
    return n1 * n;
}


int factIter(int n) {
    int fact = 1;
    for (int i = 1; i <= n; ++i) {
        fact = fact * i;
    }
    return fact;
}
```

Exercise 2. [15 Marks] Consider a 16-bit computer with a simplified memory hierarchy. This hierarchy contains a single cache and an unbounded backing memory. The cache is 2-way set associative, 4-byte cache lines, and a capacity of 32 bytes. Consider also the following sequence of memory *word addresses*.

$$\boxed{8,\ 3,\ 9,\ 7,\ 15,\ 20,\ 22,\ 2,\ 6,\ 0}$$

(a) Determine, in binary notation, the set index and block offset for each address in the above sequence. Include the *byte offset* as part of the *block offset*. Assume the cache is initially empty.

During the sequence of address accesses above, determine if each reference results in a cache hit or a cache miss. If the reference results in a cache miss, which type of cache miss occurs (cold, conflict, or capacity). Use the below table to help answer this question.

Address	Index	Block Offset	Hit or Miss	Type of Miss
8				
3				
9				
7				
15				
20				
22				
2				
6				
0				

(b) Create a table which resembles this cache's configuration. Fill that table such that it corresponds to the cache's contents after all addresses in the above sequence have been referenced. (See "3350-L4-CacheExample.pdf").

Exercise 3. [30 Marks] In this exercise, we will examine cache capacity and its effect on performance. For simplicity, let assume consider only *data cache*. That is, instructions are not stored in the caches. Recall cache access time is related to its capacity. Consider that accessing main memory requires 100ns and that, in a particular program, 42% of instructions incur a data access.

For two different processors executing this program, we have two different L1 caches, attached to processors P1 and P2, respectively.

	L1 size	L1 Miss Rate	L1 Hit Time
P1	16 KB	3.6%	1.26 ns
P2	32 KB	3.1%	2.17ns

(a) What is the AMAT for P1 and P2 assuming no other levels of cache?

(b) Assuming an ideal CPI of 2.0 for both processors, and where the L1 hit time determines the cycle time, what is the CPI_{stall} for P1 and P2? Which processor is faster at executing this particular program?

Now consider the addition of an L2 cache to P1 with the following characteristics. The data from the previous table still holds.

L2 size	L2 Miss Rate	L2 Hit Time
8 MB	48%	26.24 ns

(c) What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

(d) Assuming an ideal CPI of 2.0 and where the L1 hit time determines the cycle time, what is the CPI_{stall} for P1 with the addition of an L2 cache?

(e) Which processor is faster, now that P1 has an L2 cache? If P1 is faster, what miss rate would P1 need in its L1 cache to match P2's performance? If P2 is faster, what miss rate would P2 need in its L1 cache to match P1's performance?

Exercise 4. [20 Marks] Consider a 64-bit computer with a simplified memory hierarchy. This hierarchy contains a single cache and an *unbounded* backing memory. The cache has the following characteristics:

- Direct-Mapped, Write-through, Write allocate.
- Cache blocks are 4 words each.
- The cache has 256 sets.

(a) Consider the following code fragment in the C programming language to be run on the described computer. Assume that: program instructions are not stored in cache, arrays are cache-aligned (the beginning of the array aligns with the beginning of a cache line), `int`s are 32 bits, and all other variables are stored only in registers.

```
int N = 32768;
int A[N];
for (int i = 0; i < N; i += 2) {
    A[i] = A[i+1];
}
```

Determine the following:

(i) The number of cache misses.

(ii) The cache miss rate.

(iii) The type of cache misses which occur.

(b) Consider the following code fragment in the C programming language to be run on the described computer. Assume that: program instructions are not stored in cache, arrays are cache-aligned (the beginning of the array aligns with the beginning of a cache line), `int`s are 32 bits, and all other variables are stored only in registers.

```
int N = 32768;
int A[N];
int B[N];
for (int i = 0; i < N; ++i) {
    B[i] = A[i];
}
```

Determine the following:

(i) The number of cache misses.

(ii) The cache miss rate.

(iii) The type of cache misses which occur.

Exercise 5. [20 Marks] The Intel Core i7-8750H processor (more details here) has the following characteristics, taken from `/proc/cpuinfo`:

```
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
stepping        : 10
microcode       : 0xea
cpu MHz         : 2200.000
L1 cache size   : 32 KB
L2 cache size   : 256 KB
L3 cache size   : 9216 KB
physical id     : 0
siblings        : 12
core id         : 0
cpu cores       : 6
{...}
clflush size    : 64
cache line size : 64
```

Consider the following two functions `Normalize1` and `Normalize2` which take in a positive NxN matrix of `doubles` and *normalizes* its entries to the range $[0, 1]$. A program implementing these functions is available on OWL as `Normalize.c`.

After those two code segmenets, data is presented which was collected using the `perf` utility. This data shows runtime performance metrics of these functions executing on the Intel Core i7-8750H processor for various data sizes. In this data:

- "`Normalize.bin 1 ...`" executes the function `Normalize1`;
- "`Normalize.bin 2 ...`" executes the function `Normalize2`;
- the second command-line argument is the size N of the matrix;
- `LLC-loads` means "Last Level Cache loads", the number of accesses to L3;
- `LLC-load-misses` means "Last Level Cache misses", the number of L3 cache misses;
- `cpu-cycles` is the number of CPU cycles elapsed during programing execution.

Using the knowledge learned so far in this course, the specification of the i7-8750H processor, the code fragments, and the `perf` data, answer the following questions.

(a) Why is the runtime execution of `Normalize1` faster than `Normalize2`? The values of which performance metrics from the `perf` data support your claims?

(b) Consider the miss rates of `Normalize1`. The miss rate drastically increases for values of N larger than 512. Explain why the increase occurs at this particular value of N. Give a reason why this increase is not a sharp "jump" but rather has an intermediate effect at $N = 1024$.

(c) Consider the miss rates of `Normalize2`. The miss rate starts quite low but quickly increases for increasing values of N. Disucss why the miss rates for $N = 256$ and $N = 512$ are misleading for describing the actual data locality of `Normalize2`. Which additional performance metrics would you record to get a more precise understanding of the data locality of `Normalize2`? Assume `perf` is capable of reporting any possible hardware event.

```c
void Normalize1(double* A, int N) {
    double t, min = (double) RAND_MAX, max = 0.0;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            t = A[i*N + j];
            if (t < min) min = t;
            if (t > max) max = t;
        }
    }

    double frac = 1 / (max - min);
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            A[i*N + j] = (A[i*N + j] - min) * frac;
        }
    }
}
```

```c
void Normalize2(double* A, int N) {
    double t, min = (double) RAND_MAX, max = 0.0;
    for (int j = 0; j < N; ++j) {
        for (int i = 0; i < N; ++i) {
            t = A[i*N + j];
            if (t < min) min = t;
            if (t > max) max = t;
        }
    }

    double frac = max - min;
    for (int j = 0; j < N; ++j) {
        for (int i = 0; i < N; ++i) {
            A[i*N + j] = (A[i*N + j] - min) / frac;
        }
    }
}
```

Runtime performance of `Normalize1`.

```
Performance counter stats for './Normalize.bin 1 256':

4,802,912      cpu-cycles
4,538          LLC-loads
425            LLC-load-misses           #    9.37% of all LL-cache accesses

0.002047926 seconds time elapsed


Performance counter stats for './Normalize.bin 1 512':

16,577,772      cpu-cycles
5,409           LLC-loads
668             LLC-load-misses           #   12.35% of all LL-cache accesses

0.004451773 seconds time elapsed


Performance counter stats for './Normalize.bin 1 1024':

64,124,090       cpu-cycles
14,016           LLC-loads
6,100            LLC-load-misses           #   43.52% of all LL-cache accesses

0.016784636 seconds time elapsed


Performance counter stats for './Normalize.bin 1 1536':

143,240,802      cpu-cycles
24,864           LLC-loads
14,732           LLC-load-misses           #   59.25% of all LL-cache accesses

0.038767478 seconds time elapsed


Performance counter stats for './Normalize.bin 1 2048':

254,735,648      cpu-cycles
40,122           LLC-loads
24,405           LLC-load-misses           #   60.83% of all LL-cache accesses

0.065211502 seconds time elapsed


Performance counter stats for './Normalize.bin 1 2560':

399,380,980      cpu-cycles
73,006           LLC-loads
50,260           LLC-load-misses           #   68.84% of all LL-cache accesses

0.102457893 seconds time elapsed
```

Runtime performance of `Normalize2`.

```
Performance counter stats for './Normalize.bin 2 256':

5,870,023      cpu-cycles
117,911        LLC-loads
287            LLC-load-misses           #    0.24% of all LL-cache accesses

0.001692968 seconds time elapsed


Performance counter stats for './Normalize.bin 2 512':

22,547,539      cpu-cycles
532,221         LLC-loads
7,504           LLC-load-misses           #    1.41% of all LL-cache accesses

0.007333935 seconds time elapsed


Performance counter stats for './Normalize.bin 2 1024':

121,908,975      cpu-cycles
2,047,279        LLC-loads
372,388          LLC-load-misses           #   18.19% of all LL-cache accesses

0.031579656 seconds time elapsed


Performance counter stats for './Normalize.bin 2 1536':

292,392,531      cpu-cycles
4,528,106        LLC-loads
1,384,196        LLC-load-misses           #   30.57% of all LL-cache accesses

0.074422344 seconds time elapsed


Performance counter stats for './Normalize.bin 2 2048':

694,112,174      cpu-cycles
8,338,987        LLC-loads
6,798,261        LLC-load-misses           #   81.52% of all LL-cache accesses

0.173902216 seconds time elapsed


Performance counter stats for './Normalize.bin 2 2560':

1,025,207,852      cpu-cycles
12,349,636         LLC-loads
10,425,562         LLC-load-misses         #   84.42% of all LL-cache accesses

0.258983051 seconds time elapsed
```