# Assignment 3 (10%)

**CompSys304 S2 2019**

Maryam Hemmati

**Due: Friday 18 October, 11:59 pm**

**Submission (1%)**

- Due: Friday 18 October, 11:59 pm
- Late penalties: up to 24h late: -20% of achieved marks, 24h to 48h late: -30% of achieved marks, more than 48h late: 0 marks
- Upload your submission as ***one zip file*** on **Canvas**

The zip file should contain:

1. One pdf file named as ***assignment3_answer.pdf*** which includes your answers.
2. One folder named as ***experiment*** which includes:
    a. ***Makefile***
    b. ***cachetest1.c***
    c. ***cachetest2.c***

***There must not be other files in the zip file than the ones specified.***
The second question of this assignment can be done on any computer system you have access to. Use the assignment.c and Makefile templates from Canvas to write your programs. *Adhering correctly to these specifications gives you 1% (out of 10%)!*

## 1. Cache parameters and performance (4%)

A system has a word size of q = 64 bits and the (physical) address size of p = 40.

A. Consider the system uses an L1 unified cache for data and instruction which has 512 lines in total and a block (line) size of 128 bytes. Find out the number of bits used as byte offset (*b*), block offset (*m*), index (*n*), and tag (*t*) for two different cases:
    i. The cache is organised as direct-mapped cache.
    ii. The cache is organised as 2-way set-associative cache.

B. Consider the system uses split data and instruction caches. Access times and miss rates for an average program are given in the table below. The miss rates are the local miss rates, i.e., the percentage of misses for the requests coming through to the specific level.

| | Access time (cycles) | Miss rate (local) |
|---|---|---|
| L1 Instruction Cache | 4 | 2% |
| L1 Data Cache | 4 | 15% |
| Main memory | 100 | - |

Calculate the average memory access time (AMAT) for the system. Assume that 60% of the memory accesses are for Instructions, and the other 40% of memory accesses are for Data.

C. Now consider an L2 cache is added to the system. Access times and miss rates for an average program are given in the table below.

| | Access time (cycles) | Miss rate (local) |
|---|---|---|
| L1 Instruction Cache | 4 | 2% |
| L1 Data Cache | 4 | 15% |
| L2 | 30 | 40% |
| Main memory | 100 | - |

Calculate the average memory access time (AMAT) for the system. Assume the same instruction/data memory access mix as in B. (I. e. 60% for Instructions, and 40% for Data)

## 2. Cache experiment (5%)

Your task is to write a short C program that measures the access time differences of the caches in your system. To perform the measurement write a short program that consists of a loop with *N* iterations over an integer array *a* of size *N*, where you sum the value of each element of *a*. The execution of this loop is repeated *M* times, where *M* needs to be a large number. Measure the time it takes to execute this and divide by the total number of iterations, i.e. (*M* x *N*). Repeat this measurement for many different sizes of *N* (also changing the size of the array a), ranging from smaller than half the L1 cache to much bigger than L3 cache. Do these measurements in two different ways:

i. Go ***linearly*** through the array a
ii. Go ***randomly*** through the array a

To achieve the two different ways and have a fair comparison, use a helper array *b* which you initialise with values from *0* to *N-1* in order. Use *b* as the index for array *a*, i.e. *a*[*b*[*i*]]. For case 1 you use b directly after initialising, in case 2 you first shuffle the values. Do this by randomly choosing two elements of *b* (using *rand()*) and swap their values. Repeat this swapping at least *N* times.

**Details**

Initialise array *a* with some values and make sure that you get the same sum in both cases. The array size and number of repeats should be input parameters of your program, see template. The runtime of what you measure must be at least 400ms, otherwise your measurements will not be precise. Seconds is better.

Document the following in your pdf

- Name of your processor, e.g. Intel Core i7 7500U (use e.g. from */proc/cpuinfo*)
- Cache sizes of your system. Determine the cache sizes of your system for example using *getconf -a*, or *cat/proc/cpuinfo* or any other method.
- Table of measured time per iteration for each case. An example for such a table is this:

| *N* | size of *a* | time per iteration/ns case 1 | time per iteration/ns case 2 |
|---|---|---|---|
| 2048 | 8KiB | 5 | 25 |
| … | | … | … |
| 16Mi | 64MiB | 70 | 300 |

- Chart (time-per-iteration over size of a) of the previous table visualising the behaviour of the two cases.
- Briefly explain the differences (or similarities) of the results. Explain changes due to the different sizes of *a* and differences between the two cases.

**Notes**

To avoid unexpected results in the measurements, it is important that the computed results of the measured code are used. Compilers perform an optimization called ’dead code elimination’. If a calculation is done in your code but never used, the compiler might completely drop the code. For your assignment that means if you calculate the sum, but never use it, the compiler generates no code for it. To avoid this, have a *printf* of the sum after the timing. Also, when timing your code, make sure that nothing else is running on your system.