

1.1

cache	Cache line size	Total size	associativity(ways)	size	Number of sets
L1d (data)	64 B	192 KB	8	32 KB	$32768/(64*8) = 64$
L1i (instruction)	64 B	192 KB	NA	32 KB	NA
L2	64 B	3 MB	8	512 KB	$524288/(64*8) = 1024$
L3	64 B	8 MB	16	4 MB	$4194304/(64*16) = 4096$

1.2

```
#include <stdio.h>
#include <stdint.h>
#include <x86intrin.h>
#include <stdlib.h>

#define NUM_SAMPLES 100000

uint64_t measure_latency(volatile int *addr, int flush_cache) {
    uint64_t start, end;

    if (flush_cache) {
        _mm_clflush((void *)addr);
    }

    _mm_lfence();
    start = __rdtsc();
    _mm_lfence();
    int temp = *addr;
    _mm_lfence();
    end = __rdtsc();
    _mm_lfence();

    return end - start;
}
```

```

int main() {
    int *array = (int *)malloc(sizeof(int));
    uint64_t total_cache = 0, total_dram = 0;

    volatile int dummy = *array;

    for (int i = 0; i < NUM_SAMPLES; ++i) {
        total_cache += measure_latency(array, 0);
        total_dram += measure_latency(array, 1);
    }

    printf("Average Cache Access Latency : %lf cycles\n", total_cache /
(double)NUM_SAMPLES);
    printf("Average DRAM Access Latency : %lf cycles\n", total_dram /
(double)NUM_SAMPLES);

    free(array);
    return 0;
}

```

Measured latency

output-

Average Cache Access Latency : 94.252830 cycles

Average DRAM Access Latency : 961.722300 cycles

Memory Type	Measured Latency(cycles)
Cache	94.252830
DRAM	961.722300

measured the latency using `rdtsc` and `lfence` for accurate timestamping. Cache hits showed ~94 cycles while cache flushes followed by access (to simulate DRAM) showed ~961 cycles.

```

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <x86intrin.h>
#include <emmintrin.h>

```

```

#define ITERATIONS 1000
#define ARRAY_SIZE (1024 * 1024 * 8) // 8 MB

uint64_t measure_access_time(volatile int *addr, int flush) {
    uint64_t start, end;
    if (flush) _mm_clflush((void *)addr);
    _mm_lfence();
    start = __rdtsc();
    *addr;
    _mm_lfence();
    end = __rdtsc();
    return end - start;
}

int main() {
    volatile int *array = malloc(sizeof(int) * ARRAY_SIZE);
    if (!array) {
        perror("malloc failed");
        return 1;
    }

    for (int i = 0; i < ARRAY_SIZE; i += 64) array[i] = i;

    FILE *fp = fopen("latencies.csv", "w");
    fprintf(fp, "type,latency\n");

    for (int i = 0; i < ITERATIONS; ++i) {
        uint64_t t = measure_access_time(&array[0], 0);
        fprintf(fp, "cache,%lu\n", t);
    }

    for (int i = 0; i < ITERATIONS; ++i) {
        uint64_t t = measure_access_time(&array[0], 1);
        fprintf(fp, "dram,%lu\n", t);
    }


    fclose(fp);
    free((void *)array);

    printf("Latency data written to latencies.csv\n");
    return 0;
}

```

Generated latencies.csv file

```
from google.colab import files
uploaded = files.upload()
```



 latencies.csv

- **latencies.csv**(text/csv) - 18640 bytes, last modified: 14/06/2025 - 100% done


Saving latencies.csv to latencies (1).csv

```
import pandas as pd
```

```
df = pd.read_csv("latencies.csv")
df.head()
```

	type	latency
0	cache	1050
1	cache	63
2	cache	63
3	cache	63
4	cache	63



Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x="latency", hue="type", element="step", stat="count", bins=50, palette="Set2")
plt.title("Access Latency Distribution (Cache vs DRAM)")
plt.xlabel("Latency (CPU cycles)")
plt.ylabel("Number of Samples")
plt.grid(True)
plt.show()
```

