CS311 (Computer Architecture Lab) **Assignment-06**

Vidit Parikh [MC23BT010]

April 13, 2025

1 Introduction

The goal of this assignment was to simulate the addition of caches for a better memory access in the stages which require interaction with the memory. This is done by implementing the following changes:

- One cache (`instCache`) was added between the Instruction Fetch (IF) stage and the main memory. This cache is known as: `L1i-cache` or the "Level-1 Instruction Cache".
- One more kind of cache (`dataCache`) was added between the Memory Access (MA) stage and the main memory. This cache is known as: `L1d-cache` or the "Level 1 data cache".

2 Cache Configurations

This is in line with the specifications as mentioned in the assignment problem statement.

Cache Size	**16B**	**128B**	**512B**	**1kB**
Latency	1 cycle	2 cycles	3 cycles	4 cycles
Line Size	4B			
Associativity	2			
Write Policy	Write Through			

Table 1: Cache Configurations

3 Observations & Tabulations

Now, in this section, we shall consider two cases — one, when the size of the data cache is fixed at $1kB(1024B)$, while we vary the instruction cache values to note various observations upon running the simulator program.

3.1 Varying Instruction Cache

We fix the size of the `L1d-cache` at 1kB. Then, we vary the size of the L1i-cache from 16B $\rightarrow$ 128B $\rightarrow$ 512B $\rightarrow$ 1024B (1kB). The latency is supposed to be changed accordingly. Now, we study the performance (IPC - instructions per cycle) and plot the graph as attached.

Name of Program	**IPC**				
	No Cache	**16B**	**128B**	**512B**	**1024B**
`descending.asm`	0.02493	0.02293	0.08848	0.07804	0.06977
`evenorodd.asm`	0.02439	0.02238	0.02158	0.02083	0.02013
`fibonacci.asm`	0.02482	0.02291	0.05864	0.05313	0.04850
`prime.asm`	0.02463	0.02375	0.05022	0.04576	0.04202
`palindrome.asm`	0.02462	0.02315	0.06947	0.06113	0.05458

Table 2: Instruction Per Cycle (IPC) across different L1i-cache sizes

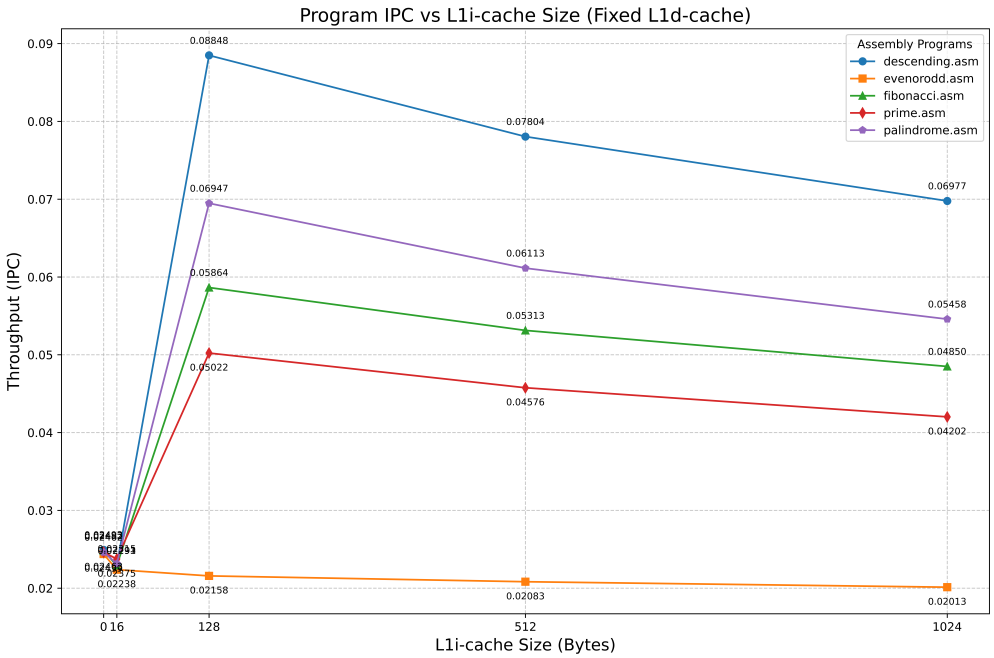

Figure 1: Graph for comparing the varying L1i-cache

3.2 Varying Data Cache

We now fix the size of the `L1i-cache` at 1kB. Then, we vary the size of the L1d-cache from 16B → 128B → 512B → 1024B (1kB). The latency is supposed to be changed accordingly. Now, we study the performance (IPC - instructions per cycle) and plot the graph as attached.

Name of Program	**IPC**				
	No Cache	**16B**	**128B**	**512B**	**1024B**
`descending.asm`	0.02493	0.06490	0.07075	0.07025	0.06977
`evenorodd.asm`	0.02439	0.02013	0.02013	0.02013	0.02013
`fibonacci.asm`	0.02482	0.04903	0.04885	0.04867	0.04850
`prime.asm`	0.02463	0.04202	0.04202	0.04202	0.04202
`palindrome.asm`	0.02462	0.05458	0.05458	0.05458	0.05458

Table 3: Instruction Per Cycle (IPC) across different L1d-cache sizes

4 Observations & Conclusions

- We successfully modelled the updated latency in the simulator after implementing cache memory. The updates have been made in the InstructionFetch (IF) Stage and during the load/store process, under the MemoryAccess (MA) Stage.
- For the code based implementation, separate classes namely - `Cache.java` and `CacheLine.java` were created within `Processor.memorysystem`.
- From the graphical representation of the table, we observe that the introduction of an `L1i-cache` decreases IPC initially, followed by a sharp increase (exception being in case of `evenoradd.asm`). Additionally, IPC steadily decreases with increase in `L1i-cache` size.
- Also, it can be observed that introduction of an `L1d-cache` sharply increases IPC initially, followed by converging the IPC on a further increase in the `L1d-cache size`.

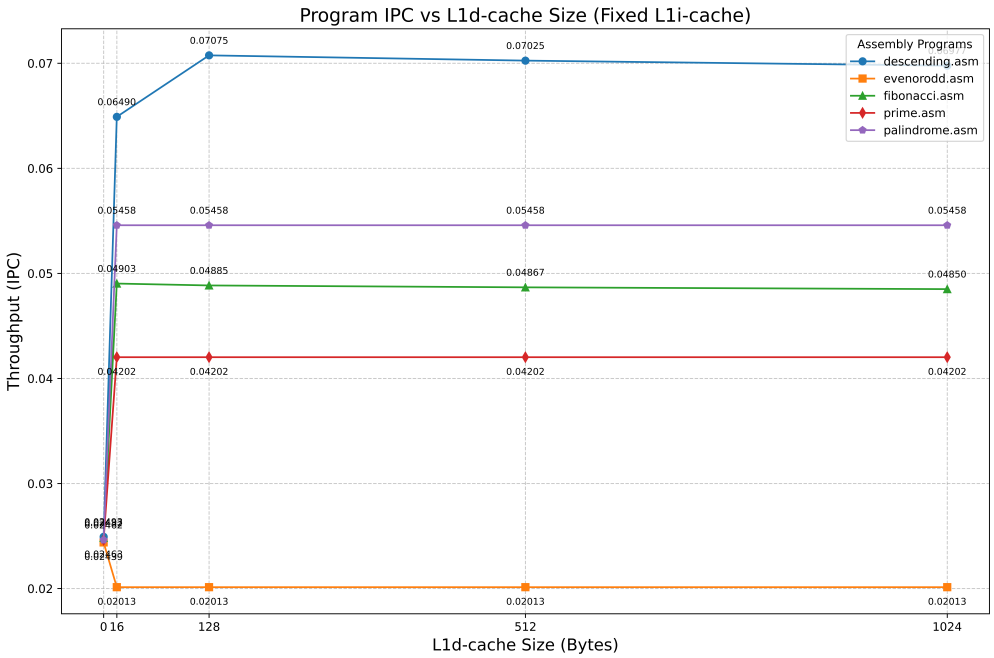

Figure 2: Graph for comparing the varying L1d-cache

- We can conclude that with a considerable amount of data, we can find an optimal value for the sie of `L1i-cache` and `L1d-cache` to maximise the throughput (IPC).

A Code for Generating Graphs

```python
import matplotlib.pyplot as plt

# Defining the data
X = [0, 16, 128, 512, 1024]
Y_descending = [0.02493, 0.02293, 0.08848, 0.07804, 0.06977]
Y_evenorodd = [0.02439, 0.02238, 0.02158, 0.02083, 0.02013]
Y_fibonacci = [0.02482, 0.02291, 0.05864, 0.05313, 0.04850]
Y_prime = [0.02463, 0.02375, 0.05022, 0.04576, 0.04202]
Y_palindrome = [0.02462, 0.02315, 0.06947, 0.06113, 0.05458]

# Initialise the figure (graph)
plt.figure(figsize=(12, 8))

# Plot each line with label
plt.plot(X, Y_descending, "-o", label="descending.asm")
plt.plot(X, Y_evenorodd, "-s", label="evenorodd.asm")
plt.plot(X, Y_fibonacci, "-^", label="fibonacci.asm")
plt.plot(X, Y_prime, "-d", label="prime.asm")
plt.plot(X, Y_palindrome, "-p", label="palindrome.asm")

# Annotate all points
def annotate_points(X, Y, offset=(0, 10)):
    for x, y in zip(X, Y):
        plt.annotate(f"{y:.5f}", xy=(x, y), textcoords="offset points",
                     xytext=offset, ha='center', fontsize=8)

annotate_points(X, Y_descending, offset=(0, 10))
annotate_points(X, Y_evenorodd, offset=(0, -15))
annotate_points(X, Y_fibonacci, offset=(0, 10))
annotate_points(X, Y_prime, offset=(0, -15))
annotate_points(X, Y_palindrome, offset=(0, 10))

# Axis labels and title
plt.title("Program IPC vs L1i-cache Size (Fixed L1d-cache)", fontsize=16)
plt.xlabel("L1i-cache Size (Bytes)", fontsize=14)
plt.ylabel("Throughput (IPC)", fontsize=14)
plt.xticks(X)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(title="Assembly Programs", fontsize=10)
plt.tight_layout()

# Save the figure
plt.savefig("case1.png", dpi=1000)

plt.show()
```

Listing 1: Code for graph for varying l1i-cache (l1d-constant