1 Feb 24, 2025: Caches I, AMAT

1.1 Memory, Storage, and Prefixes

- DRAM: Dynamic Random Access Memory
  - Volatile (data is lost if power supply interrupted)
  - Far away from the processor.
- SRAM: Static Random Access Memory
  - Non-volatile
- Disk:
  - Outdated; used to be physical disks (eg. CDs) rotating inside a computer.
- Cache: Small memory area close to the processor.

1.2 Memory Hierarchy

- We want to balance ease/convenience of accessing stored data vs. capacity to store large amounts of data.
- The way we do this is via the **memory hierarchy** - we have different types of memory storage at different "distances" from the CPU, with data relevant to current task being stored closer.
- The memory hierarchy makes the processor seem like it has a very large and fast memory.
- The hierarchy makes it seem fast.
- What makes it seem large? **Principle of Locality:** we cache the "right data" in higher levels (higher being closer to the CPU in the pyramid-diagram of the hierarchy)

1.3 Memory Cache

- The mismatch between the CPU and memory (like, full on memory like SDD or HDD drives) cause slowdowns. To solve this, we add smaller memory units closer to the processor, which are referred to as **Memory Cache**.
- These are usually on the same chip as the CPU.
- They're faster but more expensive than DRAM memory.
- The way the cache works is almost like a photocopier. If we imagine the main memory as being a library with books, then the cache is like a photocopier which contains copies of a few pages.
- Most processors have different caches for instructions vs. for data.
- The Cache consists of smaller units called **Blocks**, which can be thought of as paragraphs on a page within our library analogy.

Memory access with and without cache

Without Cache, we have to access main memory every time. If we do have Cache then we first check if the required memory blocks are already stored in Cache; if not then we pull the data from main memory. **(Please see the slides comparing memory Access with and without Cache for more info.)**

1.4 How does cache access some block from main memory?

It uses **Cache Controller Hardware**, which is also a processor. However, it is not as general purpose as a CPU. It's specialized and optimized for memory collection.

Note. *The dollar sign $ is used to denote Cache (pun intended).*

- Please take a look at the scientific notation and units for memory in the lecture slides.
- Also please take a look at the Cache-size Demo towards the end of the lecture.

1.5 AMAT: Average Memory Access Time

- So far we've spoken about things in terms of spatial measures like "distances". What about temporal measures?
- AMAT is one such measure of "temporal locality", and is the first real performance metric discussed in this class.
- When the CPU asks for some data from the Cache, there are two things that can happen: **Cache Hit** and **Cache Miss**.

Cache Hit

The data being looked for is already in the cache; it is then retrieved from the cache into the processor.

We can quantify our system's performance using two metrics:

1. **Hit Rate:** Fraction of access that hit in the cache i.e. the probability of out attempt to access memory directly from the cache being successful.
2. **Hit time:** the time taken to access cache memory, including tag comparison **(What is tag comparison? We'll see later in the course.)**

1.6 Cache Miss

The data being looked for is **not** in the cache; then we have to retrieve the data from a lower layer in the memory-hierarchy, put the data in the cache, and *then* bring the data to the processor.

Again, we can quantify our system's performance using two metrics:

1. **Miss Rate:** Just defined as $1 -$ Hit Rate (since these are both probabilities).
2. **Miss Penalty:** The time taken/latency incurred when we need to pull a block from a lower level in the memory hierarchy to the cache.

1.7 Typical scale: L1, L2 Caches

(Please see the slides to see the comparison between L1 and L2 cache; these terms weren't really properly defined in lecture)

Example: Calculation Average Memory Access Time

Consider a system without L2 Cache and assume:

- L1 Hit Time $= 1$ cycle
- L1 Miss rate $= 5\%$
- L1 Miss Penalty $= 200$ Cycles

where "Cycle" refers to a CPU Clock cycle.

In this case, what is the Average Memory Acces Time, in terms of CPU Clock Cycles? The correct answer is 11.

The idea is that

$$\text{AMAT} = \text{Hit Rate} \cdot \text{Hit Time} + \text{Miss Rate} \cdot \text{Time spent if we miss}$$

A common error people make is forgetting to account for the hit time in case we miss. The correct expression is

$$\text{Time spent if we miss} = \text{Hit time} + \text{Miss penalty}$$

Thus, we have

$$\begin{aligned}
\text{AMAT} &= \text{Hit Rate} \cdot (\text{Hit Time}) + \text{Miss Rate} \cdot (\text{Hit time} + \text{Miss penalty}) \\
&= \underbrace{(\text{Hit Rate} + \text{Miss Rate})}_{=1} \cdot \text{Hit Time} + \text{Miss Rate} \cdot \text{Miss penalty} \\
&= \text{Hit Time} + \text{Miss Rate} \cdot \text{Miss penalty} \\
&= 1 + (0.5)(200) \\
&= 11 \text{ cycles}
\end{aligned}$$

Is it possible to miss multiple times?
Yes. This is where L2 Cache becomes relevant.