Cache memory

Prof Rajeev Barua

E-A 001 -- Slide set 7

Problem with CPU organization so far

Problem with CPU organization in Slide set 3: Main memory is slow:

- 100-200 cycles per access to main memory (DRAM).

This would lead to a 100-200 cycle stall because of the IF and MEM stages, rendering the pipeline useless!

What is cache memory?

Cache memory is a way to solve this problem:

- Cache memory is a small memory that holds a frequently used subset of data of the main memory in it.
- What subset?
  - ➢ Strategies are hardware determined. Hence complex strategies not possible, since hardware has to be simple and fast.
- The cache is faster than the main memory because:
  - ➢ Built using SRAM, which is built out of logic (flip flops), and is faster than DRAM
  - ➢ Much smaller than main memory (4-5 orders of magnitude smaller than main memory ⇒ less address decoding needed.)
  - ➢ On CPU chip ⇒ tighter integration with CPU.

Two types of Locality

Locality is a property of programs.

It says that programs don’t just have random access patterns, but tend to follow certain memory access patterns.

- Leveraged by cache memories to have a high hit rate in cache despite holding only a tiny fraction of the main memory.

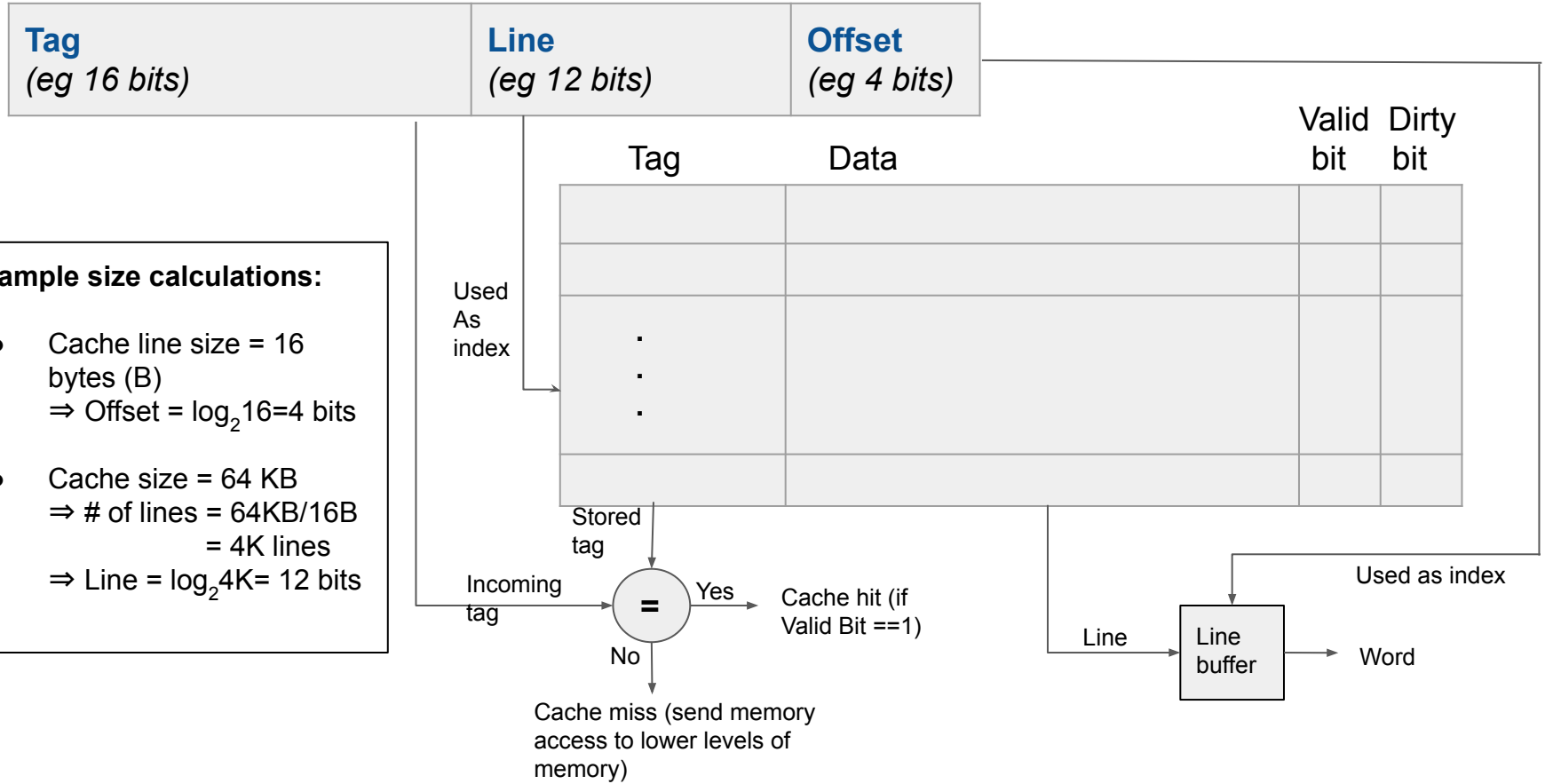
Two types of Locality

- **Temporal locality:** If word was accessed in the recent past, it will likely be accessed again soon.
  - Leveraged by caches to store whatever is accessed in the cache until it's space is needed.

- **Spatial locality:** If a word is accessed, a nearby word will likely be accessed soon.
  - Caches use CACHE LINES to take advantage of spatial locality.
  - A Cache line is a group of adjacent words (usually 16-64 bytes).

Direct-mapped caches (circuit diagram)

Address from pipeline (IF or MEM stage):



Example size calculations:

- Cache line size = 16 bytes (B)
  ⇒ Offset = $\log_2 16 = 4$ bits
- Cache size = 64 KB
  ⇒ # of lines = 64KB/16B = 4K lines
  ⇒ Line = $\log_2 4K = 12$ bits

Actions in a direct-mapped cache

1. Initially the cache is empty, so valid bits are all set to zero upon boot up of the computer.
2. When a memory address is accessed from the pipeline, its line bits are used to index into the cache.
    a. Then the incoming tag bits are compared to the stored tag at that location. If it is the same, and the valid bit is 1, it is a cache hit. Else, it is a cache miss.
3. If the access is a cache hit, we use the offset bits to access that offset in the cache line.
4. If the access is a cache miss, we first access that line from the main memory.
    a. Then we bring it into the cache. This requires evicting whatever old cache line was at that location, if that location was valid.
    b. When a cache line is evicted, it is written to the main memory if its dirty bit is 1.
5. In this way, the cache automatically stores the latest cache line to fit into each index. This promotes temporal locality.
6. The fact that the cache line is several words long promotes spatial locality.

Cache miss equation

Let: h: hit rate (fraction)

Average memory access time (AMAT)

= (cache hit time) + (cache miss penalty) * (1-h)

- *Hit rate is not included in the first term on RHS since it is paid either way.*

Example:

Cache hit time = 1 cycle

Cache miss penalty = 100 cycles (e.g., DRAM)

h = 0.95 (95% hit rate)

⇒ AMAT = 1 + 100 X (1 - 0.95) = 1 + 100 X 0.05 = 6 cycles

Unified vs. split caches

Observation: May need to fetch instructions and data at the same time in the MIPS pipeline (in IF and MEM stages, respectively)

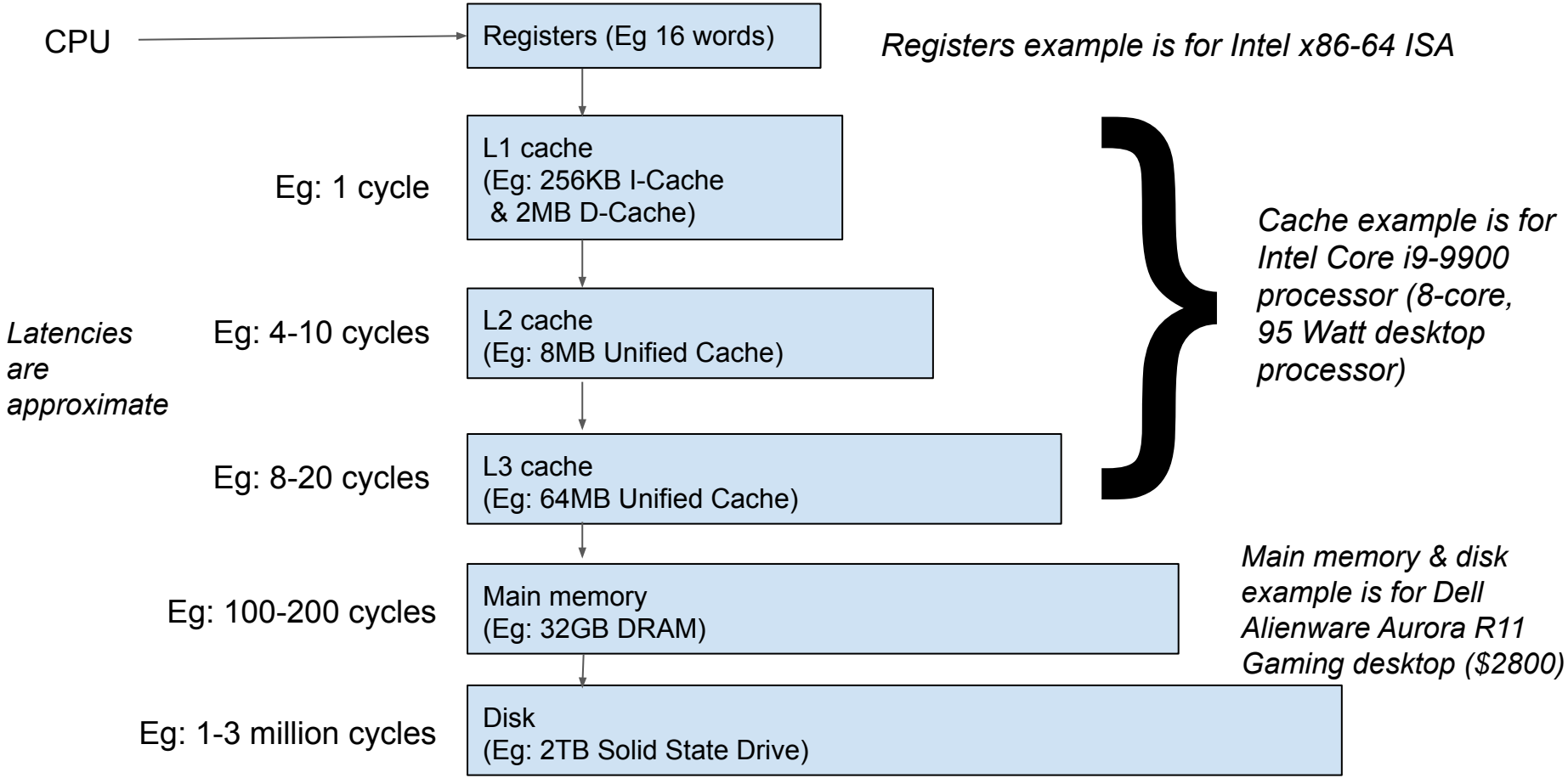
⇒ better to have separate instruction and data cache (I-Cache and D-Cache, respectively)

⇒ higher bandwidth.

An architecture with both caches above is called a split cache or Harvard architecture.

An architecture with one cache storing both instructions and data is called a unified cache otherwise.

Memory hierarchy



Calculating AMAT with multiple levels of cache

When calculating the AMAT using the cache miss equation, we must do it the following order across levels:

1. First apply it for the lowest level of cache (e.g., L3 cache.
   a. When doing so, consider that at each level the hit rate, hit time, and miss penalty numbers are all specific to that stage, and different from other stages.
   b. Usually the hit rate is the highest for the L3 cache (simply because its the biggest cache level), and lowest for the L1 cache.
   c. For the hit time, it is lowest for the L1 cache, and highest for L3 cache.
2. Then use the AMAT value calculated for the lowest level to AMAT equation at the next higher level. To do so, use `cache miss penalty = AMAT(lower level)`
   a. In this way we keep going above in the levels.

So we first calculate AMAT(L3). Then use that to calculate AMAT(L2). Which is then used to calculate AMAT(L1).

Example: AMAT calculation with multiple levels of cache

Suppose a computer has:

- L1 cache with 1 cycle hit time and 95% hit rate.
- L2 cache with 8 cycle hit time and 98% hit rate.
- Main memory with 100 cycle latency.

Q. What is the AMAT of the CPU?

A. The AMAT of the CPU is the AMAT of the L1 cache.

$AMAT_{L2}$ = Hit $time_{L2}$ + Miss $penalty_{L2}$ X (1 - hit $rate_{L2}$)

= 8 + 100 X (1 - 0.98) = 8 + 2 = 10 cycles.

$AMAT_{L1}$ = Hit $time_{L1}$ + Miss $penalty_{L1}$ X (1 - hit $rate_{L1}$) // *Miss* $penalty_{L1}$ = *AMAT* $_{L2}$

= 1 + 10 X (1 - 0.95) = 1 + 0.5 = **1.5 cycles.**