Lecture 16: September 13, 2020
Computer Architecture and Organization-II
Biplab K Sikdar

0.5 Reducing Hit Time

AMAT = *hit ratio* $\times$ *hit time* + *miss rate* $\times$ *miss penalty*.

Common techniques to reduce *hit time* are

1. Introduction of small and simple cache,
2. Avoiding time for address translation,
3. Pipelining writes,
4. Pseudo-associativity,
5. Trace cache.

0.5.1 Small and simple caches

Small cache system can reduce hit time.

Generally smaller hardware is faster as it reduces number of logic levels.

A small cache (L_1 cache) can be fitted within CPU.

On-chip cache avoids time penalty of going off chip.

In some designs, tag of cache is stored in CPU and data part of cache is off-chip.

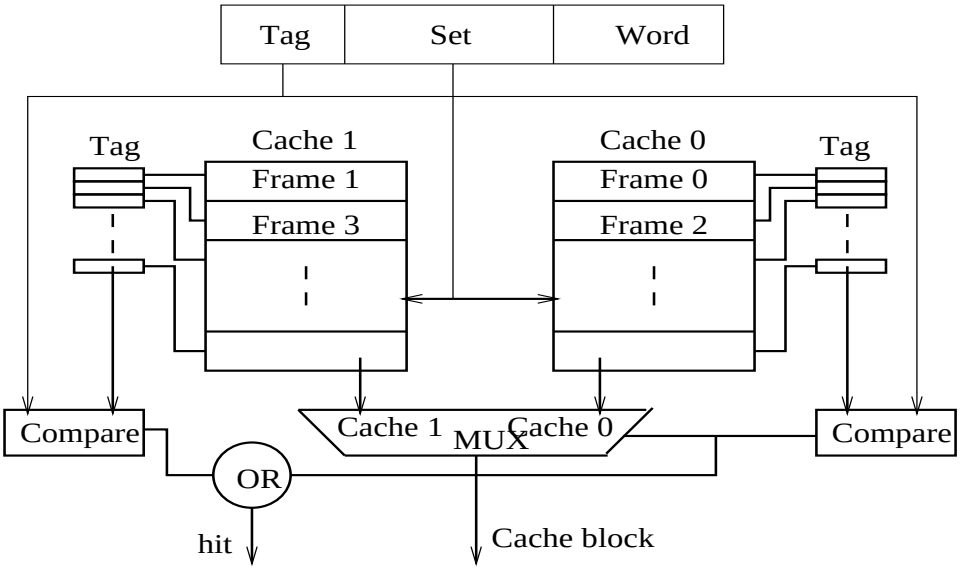
For every data access, tags are to be compared.

On-chip tag realizes faster hit.

However, on-chip cache is realizable only when cache system is small.

A small cache needs lesser number of tag bits (to be stored in CPU).

Direct mapping is simple and reduces hit time.



0.5.2 Avoiding delay in address translation

When CPU refers item x, address YY of x is generated. YY is a logical address.

For every load, store, or instruction fetch, there are two memory accesses

One for *page table* and other for fetch of information/data.

Page table and address translation mechanism (Figure 33) are to be placed

Within cache (a CPU cache) to reduce hit time.

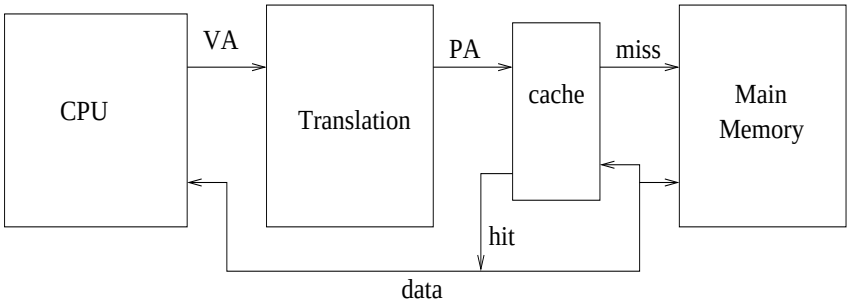


Figure 33: Cache address translation

Translation look aside buffer (TLB):

TLB contains recently used page table entries.

TLB is a small (CPU) cache - organized as fully/set associative or direct mapped.

Hardware logic with TLB is shown in Figure 34.

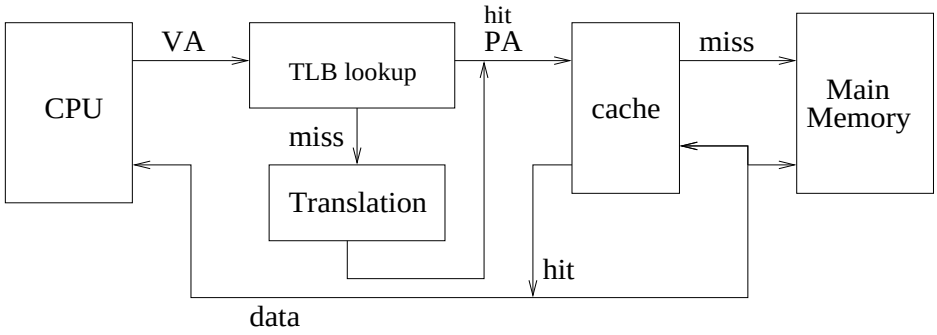


Figure 34: Cache address translation with a TLB

0.5.3 Pipelining writes

The hits are - (a) *read hit* and (b) *write hit*.

Consider direct mapping and item x, belongs to page/block p, is referred.

Mapping scheme, now, points to cache line (cache frame) B.

The next step is comparison of tags.

If reference is read, comparison of tags is done in parallel with data read from cache.

If comparison outcome is false, read block from cache is discarded.

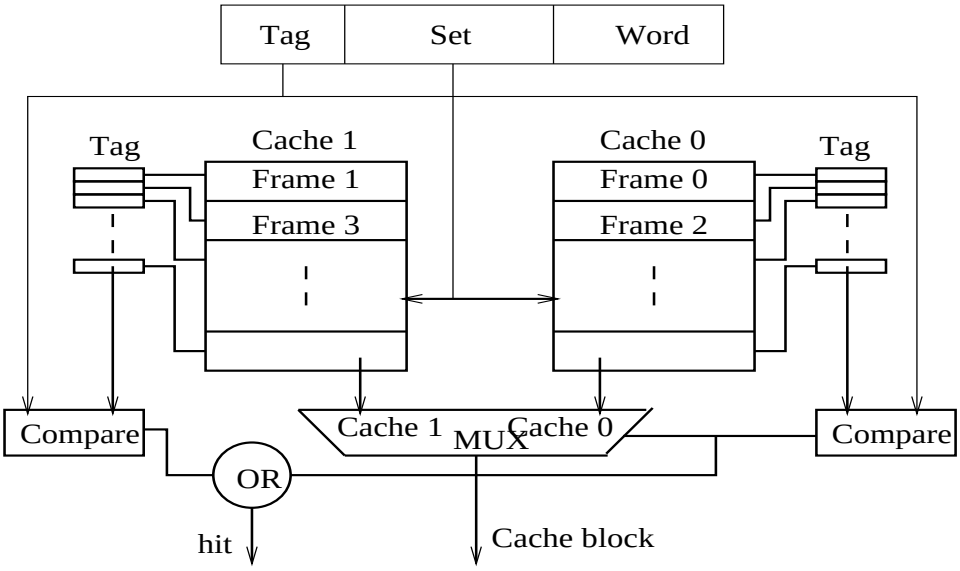


Figure 35: Set associative mapping

However, for a write, tag must be checked before data can be written.

That is, in reality, write takes longer time than reads.

Pipelined cache architecture is an effective solution for faster write hit.

Tag checking and updation (write) of cache are done in separate stages of pipeline.

At t, current write (WR_i) tag check and previous cache update (WR_{i-1}) are done.

0.5.4 Pseudo-associativity

Simplest pseudo-associative cache divides cache in two halves as in the figure.

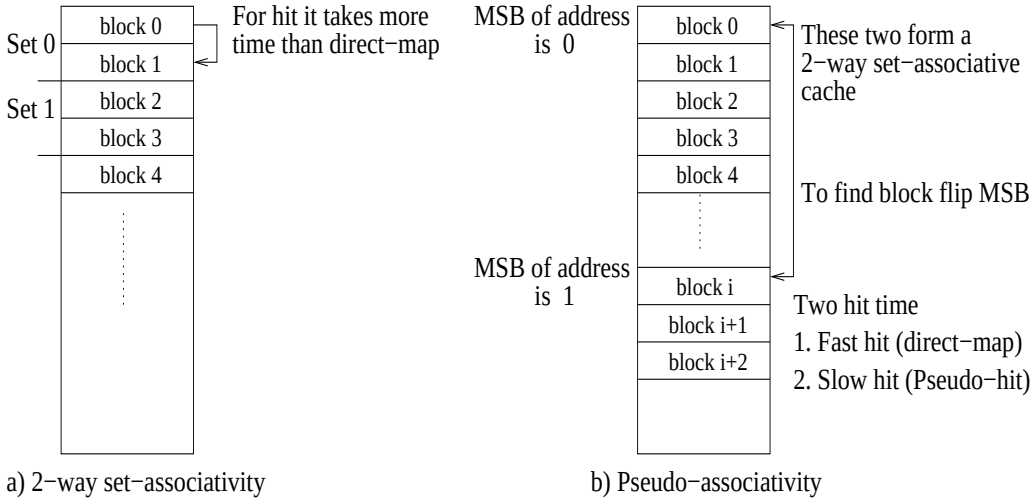


Figure 36: Pseudo-associative cache

Here, two types of hit time are found - hit and pseudo hit.

1. hit: For some references to block B, if B is found at first half,

it is a fast hit and same as that of direct mapping scheme.

2. Pseudo hit: Block B is found in other half of cache.

Pseudo hit (slow hit) time, therefore, is

delay in accessing from second half + delay in searching for B in first half.

In average, a pseudo-associative cache takes more time than direct mapped cache.

But less time than that of conventional 2-way associative mapping.

0.5.5 Trace cache

I-cache of a processor contains static instruction sequence.

On contrary, a trace cache captures dynamic traces of executed instructions.

It may contain instructions in execution path(s) of taken branches (Figure 37).

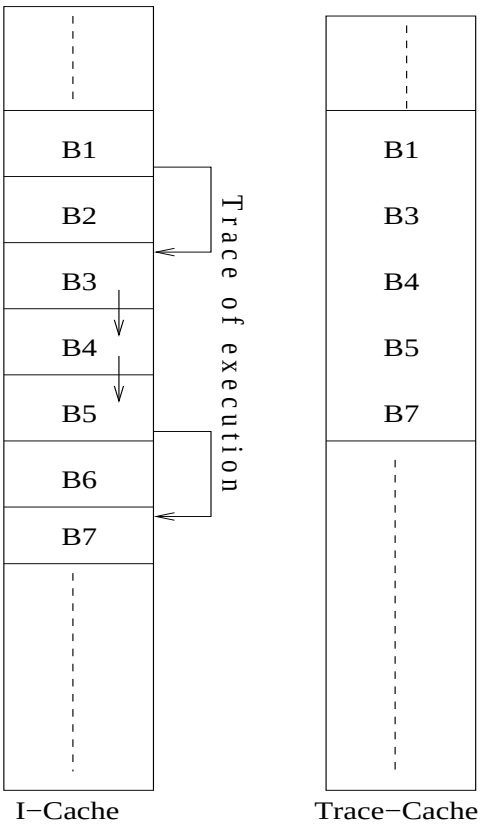


Figure 37: Trace cache

Consider the instruction stream in an I-cache (for a program) is

I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I12, I13, I14, ...

Let assume two instructions per block.

That is, B1(I1, I2), B2(I3, I4), B3(I5, I6), B4(I7, I8), B5(I9, I10), B6(I11, I12), ...

Let I2 and I9 are *taken* branch; I5 is the target address for I2 and I13 is for I9.

That is, while executing the program trace contains

I1, I2, I5, I6, I7, I8, I9, I13, I14, ...

That is, B1, B3, B4, B5, B7 ... are in a trace cache (Figure 38).

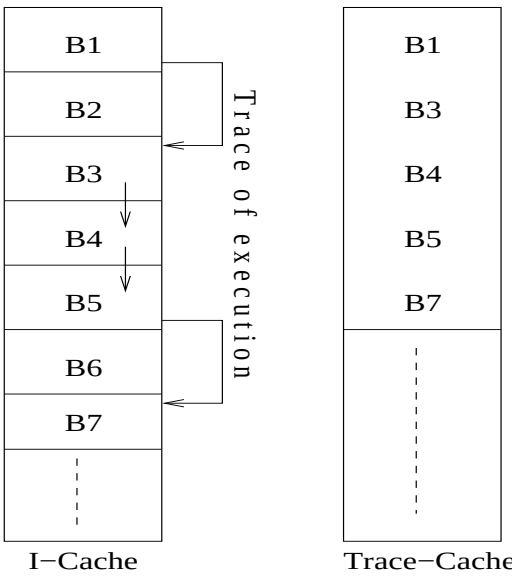


Figure 38: Trace cache example

Trace cache reduces hit time: single fetch in allows access to multiple blocks.

However, trace cache requires complicated address mapping.

An instruction may appear in multiple traces due to different branch outcomes.

0.5.6 Increasing cache bandwidth

I skip it.