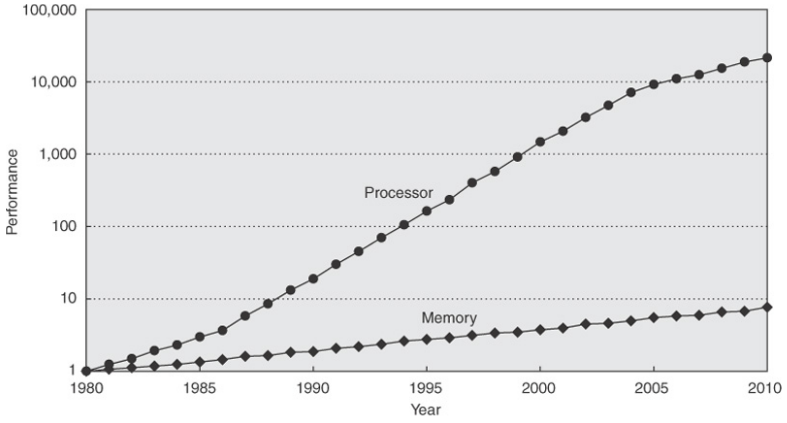
Engineering 9859
CoE Fundamentals — Computer Architecture
Memory Hierarchy Design

Dennis Peters[1]

Fall 2007

[1]Based on notes from Dr. R. Venkatesan

Speed of Memory vs CPU



Memory hierarchy – Why and How

- Memory is much slower than processor.
- Faster memories are more expensive.
- Larger memories are always slower (due to high decoding time and other reasons).
- Therefore, hierarchy of memory:
    - locate small but very fast memory (SRAM: L1 cache) very close to processor.
    - L2 cache will be larger and slower.
    - Main memory is usually GBs large and are made up of DRAMs: several nanoseconds.
    - Secondary memory is on discs (hard disk, CD, DVD), are hundreds of GBs large, and take milliseconds to access. Similar to warehouse.
    - CPU registers are the closest to CPU, but do not use memory address they have separate ids.

Principle of Locality

- Instructions and data exhibit both spatial and temporal locality:

  Temporal locality: If a particular instruction or data item is used now, there is a good chance that it will be used again in the near future.

  Spatial locality: If a particular instruction or data item is used now, there is a good chance that the instructions or data items that are located in memory immediately following or preceding this item will soon be used.

- It is a good idea to move such instruction and data items that are expected to be used soon from slow memory to fast memory (cache).
- This is prediction, and therefore will not always be correct — depends on the extent of locality.

Example: Speedup due to cache

Assume:

- 1 GHz processor ($\tau = 10^{-9}$),
- CPI $= 1$
- 10 ns memory
- 35% of executed instructions are load or store.
- Application runs 1 billion (10^9) instructions.
- Memory accesses $=$ 1 per instruction $+$ 1 for each load/store.
- Execution time $= (1 + 1.35 * 10) * 10^9 * 10^{-9} = 14.5$ s
- What if all instructions & data are stored in a perfect cache that operates within one clock cycle?
- Execution time $= 10^9 * 10^{-9} = 1$ s.

Example (cont'd)

- Now assume cache hit rate of 90%
- Execution time $= (1 + 1.35 * 0.1 * 10) = 2.35$ s
- Typical cache hit rates are
  - 95–99% for instructions.
  - 75–90% for data.
- How do we design a better cache (or caches)?

von Neumann Architecture

- Memory holds instructions (in a sequence) and data,
- Memory items are not distinguished based on their contents — any memory item is a string of bits.
- Most modern processors have instruction pipelines.
- Instruction storage exhibits stronger locality.

∴ we usually split the L1 cache into instruction cache (IC) and data cache (DC) — know as the *Harvard architecture*.

- Typical IC or DC size is 8, 16, 32, 64 or 128 kB plus overheads.

Block Placement

- Tape uses sequential access;
- RAM/ROM uses random access;
- Disc uses random/sequential access;
- Caches use associative access (i.e., uses a part of the information to find the remaining).

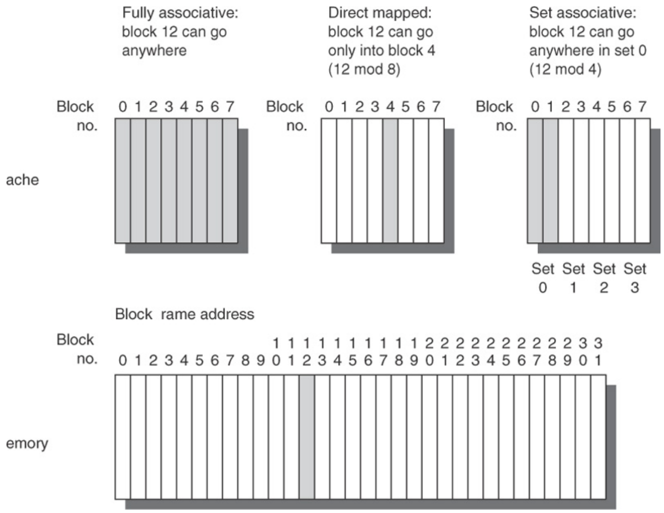
Direct mapped Each block has only one place it can appear.

Fully associative Block can be placed anywhere in cache.

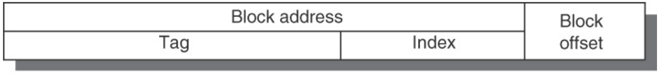
Set associative Block can be placed in a restricted set of places. *n* blocks in a set = *n way set associatve*

The first two are special cases of set associative: Direct mapped = one-way set associative, fully associative with m blocks = m-way set associative.

Associative Access



Block Identification — How is a block found?



- Each cache block frame has *tag* that associates it with memory blocks.
- *valid* bit identifies if a cache block frame contains a block from memory.
- *index* bits: select one set.
- Fully-associative — no index bits.
- Direct mapped — largest index field, but only one comparison.
- *Cache miss*: no comparison succeeds
- *Cache hit*: one comparison succeeds, block identified.
- *Block offest* bits: select byte or word — not relevant to hit/miss.

Block Replacement

Which block should be replaced on cache miss?

- Three strategies

  Random: overwrite a random block in the set.

  Least recently used (LRU): overwrite the block that has been untouched the longest.

  First in, first out: overwrite the oldest block.

- Not relevant to direct mapped. (Why?)

Write Strategy

Writing is more complicated than reading:

- block access cannot be concurrent with tag comparison
- write must only affect specified size

Need to update lower levels of cache and main memory whenever a store instruction modifies DC.

Write hit the item to be modified is in DC:

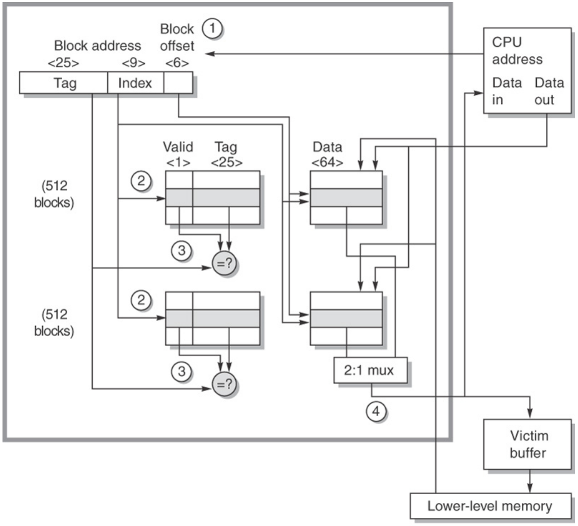
- Write through write to next level (as if no DC).
- Write back set a dirty bit and update next level before replacing this block.

Write miss the item to be modified is not in DC.

- Write allocate exploit locality, load block into DC.
- Write no-allocate don't fetch the missing block.

Usually, write through and write no-allocate policies are used together; write back and write allocate.

Example: Alpha 21264 2-way set-associative cache



Cache Size Example 1

Overheads: tag bits, valid bit, dirty bit(?)

- # address bits: 46
- block size: 64B
- 64kB 2-way IC/DC, DC uses write through, write no-allocate
- # block offset bits: $2^o = \text{block size} \Rightarrow o = 6$
- # index bits:

$$\begin{aligned} 2^i &= \#\text{blocks} = \frac{\text{cache size}}{\text{block size} * \text{assoc.}} \\ &\Rightarrow i = 9 \end{aligned}$$

(512 sets, 1024 blocks)

- tag bits: $46 - (6 + 9) = 31$
- Total IC/DC size:
  $64 * 8 * 1024\ \text{b} + (31 + 1) * 1024\ \text{b} = 64\text{kB} + 4\text{kB}$

Cache Size Example 2

- # address bits: 40
- block size: 32B
- 1MB L2 cache, 4-way set-associative uses write back & write allocate
- # block offset bits: 5
- # index bits: 13 (8192 sets, 2^{15} blocks)
- tag bits: $40 - (5 + 13) = 22$
- L2 cache size: $1\mathrm{MB} + \frac{22+1+1}{8} * 2^{15} = 1\mathrm{MB} + 96\mathrm{kB}$

Cache Performance

- Memory access time = hit time + miss rate * miss penalty
- To improve performance, i.e., reduce memory time, we need to reduce:
  - hit time,
  - miss rate
  - miss penalty.
- As L1 caches are in the critical path of instruction execution, hit time is the most important parameter.
- When one parameter is improved, others might suffer

Misses

Compulsory miss:

- block has never been in cache (during this execution of a program)
- always occurs on first access to a block

Capacity miss:

- block was in cache, but was discarded to make room for other block
- reduces with cache size

Conflict miss:

- block discarded because too many map to same set
- reduces with level of associativity.

Cache Performance Example

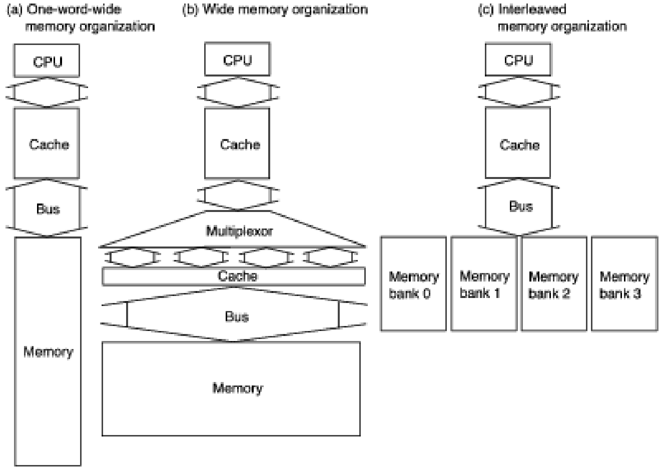
- 5 GHz pipelined processor ($\tau = 0.2$ ns);
- IC hits 98%;
- L1 read miss 10%, write miss 5%;
- 25% of all instructions are loads and 10% are writes.
- Fetching a block from L2 cache takes 40 clk. (8 ns)
- L2 misses 12%, with a penalty 25 ns (125 clk).

An L3 cache might improve the performance.

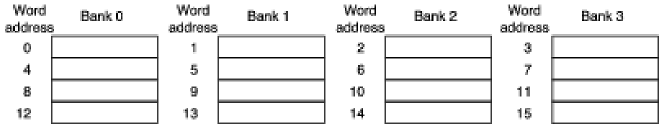
Typical Cache Structure

- Most general purpose computers today use
  - CPU chips that have on-chip IC&DC,
  - on-package L2 cache,
  - and an L3 cache on the motherboard.
- Caches are also used in the sound card, video card and with other special purpose hardware.

Main Memory Organizations



Memory Interleaving



- Access time = address transmit + decode + amplify + data transmit.
- Decode is the largest component.
- In memory interleaved system, address is transmitted and decode takes place commonly for all the banks;
- only data transmission occurs from each bank in a pipelined fashion. e.g., $0.5 + 6.0 + 0.3 + 4*0.6 = 9.2$ ns.

Virtual Memory

Relocatability programmer and compiler do not have to worry about exact location of information. Use any addresses, and leave it to loader at run time.

Security: Using extra bits, access of any information can be regulated. Controlled sharing is possible.

Different size: virtual memory can be larger than real memory. This is not the major reason nowadays.

Virtual address and Real Address $\Rightarrow$ translation

Virtual Memory Structure

Three implementation techniques:

Paging: Fixed size blocks

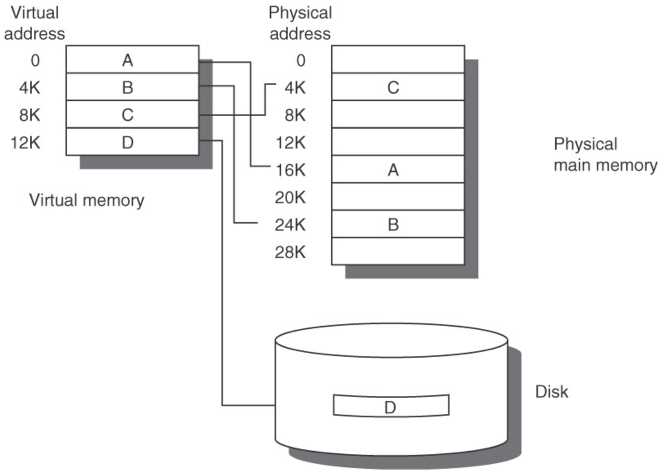
- most common
- discussed in detail here.

Segmentation: Variable size blocks

- used in early Intel processors.

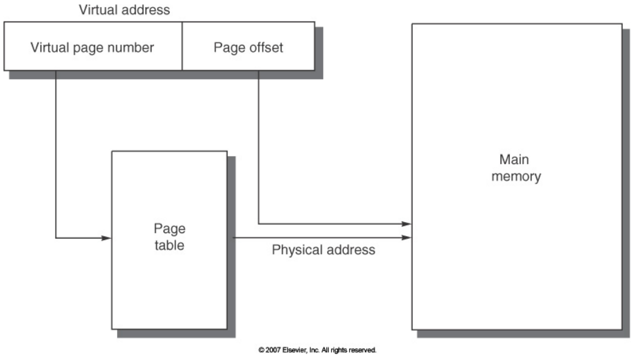
Hybrid: (paged segments) Segments are variable (integral) number of pages.

Disk & Real memories



OS uses process identifier bits (PID or ASN) and thus permits the same virtual addresses to be used by different processes.

Address Translation

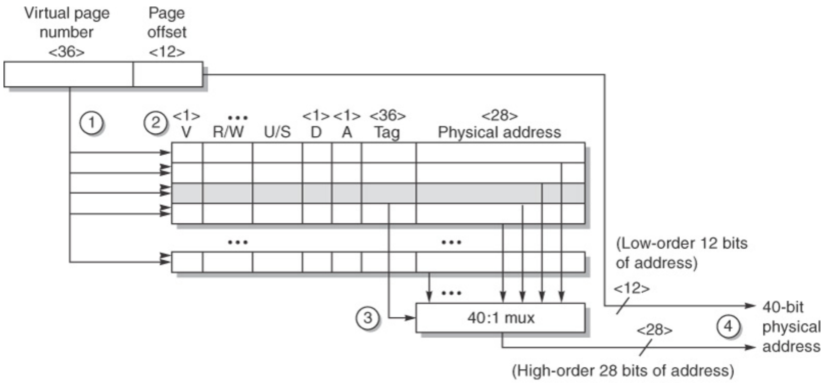


- Page table is lookup table for mapping virtual page # to physical address.
- Located in memory.
- Each instruction execution refers to memory ≈ 1.35 times, so PT is in cache, called *translation lookaside buffer* (TLB).

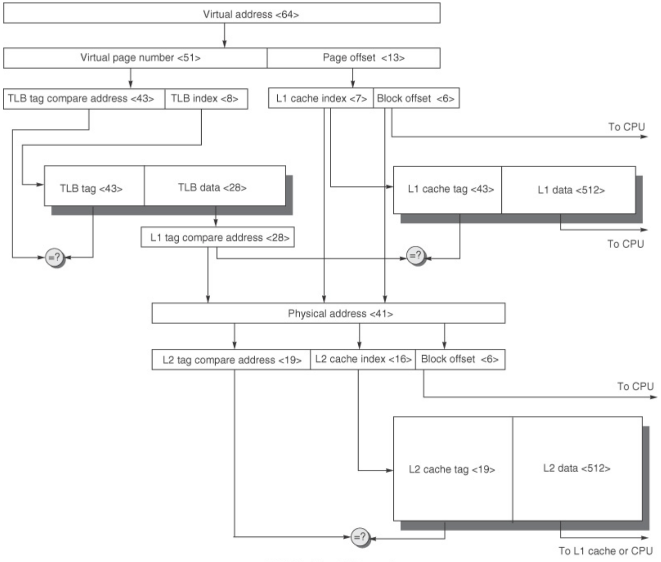
Address Translation Cache: TLB

- Usually split as Instruction TLB (ITLB) and Data TLB (DTLB).
- One level.
- When TLB misses, exception occurs and OS consults PT and updated TLB.
- TLB takes care of protection.
- ITLB: 12 – 40 entries $\Rightarrow$ 200B – 1kB.
- DTLB: 32 – 64 entries.
- Read-only, fully-associative.

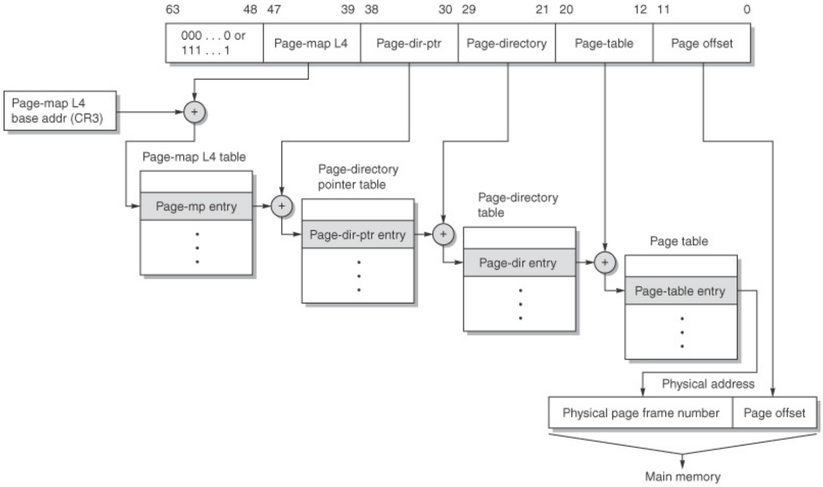
TLB Example

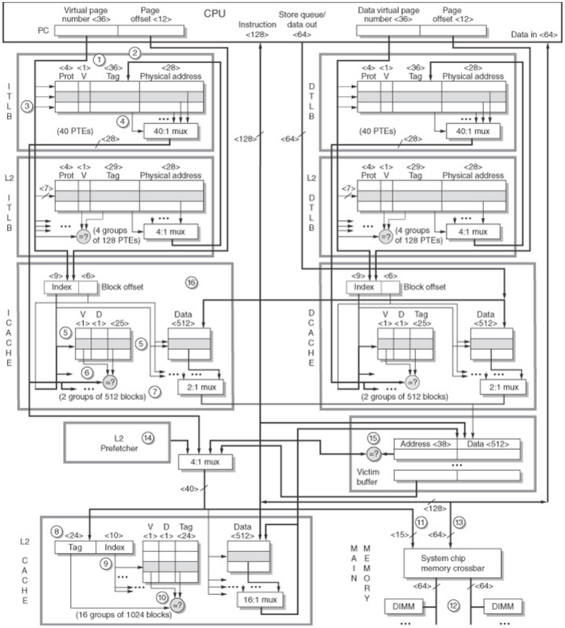

Virtual page number <36>
Page offset <12>
1
2
<1> V
R/W
U/S
<1> D
<1> A
<36> Tag
<28> Physical address
(Low-order 12 bits of address)
<12>
3
40:1 mux
<28>
4
40-bit physical address
(High-order 28 bits of address)

Memory Access: TLB & Caches



Page Table Access




CPU
PC
Virtual page number <36>
Page offset <12>
Instruction <128>
Store queue/ data out <64>
Data virtual page number <36>
Page offset <12>
Data in <64>
ITLB
DTLB
<4> Prot
<1> V
<36> Tag
<28> Physical address
(40 PTEs)
40:1 mux
<28>
L2 ITLB
L2 DTLB
<29> Tag
<7>
(4 groups of 128 PTEs)
4:1 mux
=?
<128>
<64>
ICACHE
DCACHE
<9> Index
<6> Block offset
V <1>
D <1>
Tag <25>
Data <512>
(2 groups of 512 blocks)
2:1 mux
L2 Prefetcher
4:1 mux
<40>
Victim buffer
Address <38>
Data <512>
<128>
L2 CACHE
<24> Tag
<10> Index
Tag <24>
(16 groups of 1024 blocks)
16:1 mux
MAIN MEMORY
<15>
<64>
System chip memory crossbar
DIMM
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Summary

- Memory design is extremely important: hierarchical.
- Caches improve performance due to locality present in instruction and data access.
- L1 cache design is important as it is on critical path; it is split into IC & DC if processor is pipelined.
- L2 cache (mixed) will be larger and slower than L1;
- L3 cache will be larger and slower than L2, and
- main memory will be larger and slower than L3.
- Cache performance suffers when store is performed.
- Virtual memory is used for relocatability & protection.
- Page Tables are used for virtual to real page address translation
- TLBs are caches for PT.