**Cache**


**Why does cache work?**

Based on principles of spatial and temporal locality

- **spatial** transfers of bytes adjacent to those needed now will be used in the near future
- **temporal** once in cache data will likely be reused for a while

**Measure of Cache Performance**

\[
\text{hit rate} \% = \frac{\text{cache hits}}{\text{total memory request}} \times 100\%
\]

\[
\text{miss rate} = 1 - \text{hit rate}
\]

System performance a function of **miss penalty** – time to satisfy missed cache request (may be measured in cpu cycles wasted during stall)

**Types of Misses**

- **compulsory** – first reference miss
- **conflict** – replacement due to direct or set associative mapping
- **capacity** – cannot contain all blocks

**Basic Concepts – Simplified Memory Configuration with Cache**

![Basic Concepts – Simplified Memory Configuration with Cache](image_url)

see text figures 5.1–5.5
Cache vs Local Memory

Cache vs Local Memory Content

<table>
<thead>
<tr>
<th>local memory</th>
<th>cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
</tbody>
</table>

address does not necessarily map to particular location in cache
Fundamental Reason for Cache

1. faster read cycles
2. faster, possibly decoupled, writes
3. reduced memory bus traffic

Cache Organization Techniques

**Fully Associative** – A cache structure in which every tag in the cache is compared to the tag of the datum being accessed

Advantage:

- any combination of memory locations can be cached

Disadvantages:

- low ratio of data bits to total cache bits
- need one comparator/entry
- does not take advantage of positional locality of code
Fully Associative (with increased block or line size)

Cache block or cache line – a group of data moved in or out of cache as a unit, which is anywhere from one byte to some power of 2 bytes.

Set–Associative Cache – A cache structure in which all tags in a particular set are compared with an access key in order to access an item in cache. The set may have as few as one element or as many elements as there are lines in the full cache. See 2–way set–associative cache, below.

Advantage:
- fewer comparators required
- greater data bits/total bits

Disadvantages:
- entries with same index (set i.d.) limited to number of ways associative
Ex. MC68020 On–Chip Cache Memory (Instruction Only)

- faster instruction access on hits (2 cycles vs. 3 cycles) – minimum of 33% improvement
- more bus bandwidth available to alternate bus masters
- 256 bytes organized 64 x 4 bytes

Note: MC68030 has 256 bytes of instruction and 256 bytes of data cache while the MC68040 has 4K of 4–way set–associative instruction and 4K of 4–way set–associative data cache.

- set–associative (1–way) or direct mapped
- block size = 2 words

Operation:
1. instruction access from cache and check for tag match. At the same time start memory cycle
2. If hit, valid bit checked and if ok cached instruction used. Terminate memory cycle early
3. If no match, instruction fetched for cpu from external memory and also written into cache unless freeze cache bit has been set in CACR

Note: microprocessor always uses longword alignment and updates both words in block

Control Registers:
CACR 3 (C, clear cache) 2 (CE, clear entry) 1 (F, freeze) 0 (E, enable, 0 forces misses, no fills)
CAAR 7:2 (index – invalidates entry based on index only)

Register Access through movec Rc, Rn or movec Rn,Rc
Virtual vs Physical Caches

- memory management unit (MMU) usually between CPU and cache so most caches are physical (real addresses used).
- some CPUs featuring virtual caches (logical addresses used): MC68020, MC68030, i860, R4000.
- virtual caches have synonym or virtual address alias problem – two or more virtual (logical) addresses mapping to a single physical address

Look–Through vs Look–Aside Architecture

- look–through – first access cache, if miss then access main memory
- look–aside – parallel cache and main access (e.g., MC68020 instruction caches approach). A disadvantage is that CPU main memory bus is tied up.

Write Policies

Write–Through – all writes sent to main memory

- maintains “cache coherency” (other bus masters get correct data)
- more bus traffic
- multiple masters with local cache can monitor writes and if in cache
  1. invalidate (stale data), or
  2. update

Buffered Write–Through – writes passed to buffers or FIFOs decoupling CPU write cycle from memory write cycle (both address and data buffered)

Write Back (Copy Back) – written data flagged as “dirty” or “modified” (main memory incoherent). Data written back only on line replacement or context switch resulting in less traffic on main memory bus

Cache Line Status Bits

- Line Valid – invalid implies corresponding block in memory modified
- Word Valid – bus width work word valid
- Word Dirty/Modified – data in cache modified, but not in memory
- LRU Field – used to identity “least recently used” line
- Ownership Bits – data exclusive to cache
- Access Rights – type of access space
Cache Update Policies

- for direct, no flexibility
- others can use pseudo–random or LRU
- on write misses, cache may or may not allocate – update cache
  ex. on MC68040 write–through cache controller does not “cache allocate”
- fills can be
  1. data requested last – rest of line first
  2. data requested first – rest of line follows (and this can be accessed from a line buffer)

LRU Example – 4–way set associative (like 80486 & 68040)

<table>
<thead>
<tr>
<th>L0</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B0 = 1
- If L0, B1 = 1
- else, B1 = 0
B0 = 0
- If L2, B2 = 1
- else, B2 = 0

Access example:

<table>
<thead>
<tr>
<th>use</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3</td>
<td>0</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>L1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>L0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Other CPU Examples
Z80000: tag match + word hit, LRU, single word update, write–through (if miss only update memory).
MC68030: write–through, if miss no–write allocate or write allocate selectable
Cache Coherency Issues

Stale Data Problem – memory locations for an item having different values

- multiprocessor example

1. cpu1 writes to main memory and cache
2. cpu2 writes new data to main memory
3. cpu1 reads old data from cache

- uniprocessor example

1. cpu reads data from main memory and updates cache
2. cpu writes new data to cache only
3. later, miss causes data to be replaced in cache
4. cpu reads original stale data from memory

Problem could have been avoided by using a write-through policy or dirty bit and write back at step 3.

Software Solution

Can make use of PMMU features and compiler to mark modules, data structures, and I/O as shared/noncachable, exclusive, etc.

Approaches:

- never cache shared pages
- allow read only access to shared pages
- share only during period of “exclusive” access
- use directory to keep state of each memory block
Hardware Solution

Can make cache invisible to OS for more efficient operation

Approaches:

- cacheable/noncacheable memory – make specific areas of memory non-cacheable (ex. I/O registers, ROM, etc) by combination of address decode and cache disable signal to on-chip cache
- shared cache – multiple cpu’s with common cache – doesn’t scale due to common cache access requirement
- bus snooping

Read
1. cache controller monitors memory/system bus for reads
2. if in cache, inhibit main memory access and provide data from its cache

Write
1. cache controller monitors memory/system bus for writes
2. if target address is in cache, either
3. invalidate cache entry or update it

note: requires dual ported tag RAM to minimize interference

See Fig. 5.25 (p. 295) for 80486 example.
MESI Hardware Cache Coherency Protocol – used by Intel and Motorola

Four States:

1. Modified [M]: line available only in cache. Can be accessed locally but cache controller must write back on snoop access

2. Exclusive [E]: available only to this cache and not stale. A change in cache can be done without informing other caches.

3. Shared [S]: same line in more than one cache. Must use write-through (other caches can invalidate as a result)

4. Invalid [I]: not available in cache. Reads update cache, writes cause write-through and possibly write allocate.

See Fig. 5.26 for state table and diagrams

Example:

<table>
<thead>
<tr>
<th>current state</th>
<th>action</th>
<th>next state</th>
<th>memory bus activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>snoop</td>
<td>S</td>
<td>write back</td>
</tr>
</tbody>
</table>
Cache Design

Performance Parameters:

1. Cache Performance: Function of
   - Hit Rate
   - Bandwidth
   - Miss Penalty
   Must test over a number of Programs
2. Cache Bandwidth: Function of
   - Access Time
   - Tag Compare Time
   - Propagation Delays
3. Memory Bus Utilization:
   - Cache Line Refill Time
   - Write Policy
   - Bus Architecture (Dual/Single, Buffers)
4. Main Memory Read & Write Performance
5. Cost Function of
   - Associativity (tag size)
   - Line Size

Average Memory Access Time = Hit Rate \times Hit Time + Miss Rate \times Miss Penalty

Miss Penalty = Memory Read Latency + Transfer Time + Delivery Time

= L + X + D

Time to Read First Word (Decode, Memory Access, Arbitration)

Time to Get to CPU

Time to Transfer Burst
Ex. Your are given the following:

- 1 clock cycle to send address
- 4 clock cycles to access & transfer 1 DW
- “Data Requested Last” scheme

If cache Block 1 DW

\[ Miss\ Penalty\ (1\ DW) = L + X + D = 1 + 4 + 0 = 5\ \text{clock cycles} \]

If cache Block 4 DWs

\[ Miss\ Penalty\ (4\ DW) = 4 \times (L + X) = 20\ \text{clock cycles} \]

If Burst Transfers (1 per clock)

\[ Miss\ Penalty\ (4\ DW\ Burst) = Miss\ Penalty\ for\ 1\ DW + 3\ \text{clock cycles} = 8\ \text{clock cycles} \]

**Parameters Affecting Cache**

- Cache Size & Block Size
  see Figures 5.28, 5.29, 5.30, 5.31
  need to know if CPU stalls until line fills
- Organization
  is 4–way enough?
- Hierarchy of Caches
- Fetch Policy
  prefetching?
- Replacement Policy
- Unified vs Split

**Performance Measures**

1. Specific Application
2. General Benchmarks
3. Approximate Formulas