Cache Memory

- memory hierarchy
- CPU memory request presented to first-level cache first
- if data NOT in cache, request sent to next level in hierarchy...
- and so on
Cache Hierarchy

• for a system with a first level cache and memory ONLY

\[ t_{\text{eff}} = h t_{\text{cache}} + (1-h) t_{\text{main}} \]

where

- \( t_{\text{eff}} \) = effective access time
- \( h \) = probability of a cache hit [hit rate]
- \( t_{\text{cache}} \) = cache access time
- \( t_{\text{main}} \) = main memory access time

• assuming \( t_{\text{main}} = 60\text{ns} \) and \( t_{\text{cache}} = 5\text{ns} \)

<table>
<thead>
<tr>
<th>( h )</th>
<th>( t_{\text{eff}}(\text{ns}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>5.0</td>
</tr>
<tr>
<td>0.99</td>
<td>5.6</td>
</tr>
<tr>
<td>0.98</td>
<td>6.1</td>
</tr>
<tr>
<td>0.89</td>
<td>11.1</td>
</tr>
<tr>
<td>0.50</td>
<td>32.5</td>
</tr>
<tr>
<td>0.00</td>
<td>60.0</td>
</tr>
</tbody>
</table>

• small changes in hit ratio [as \( h \to 1 \)] are amplified by \( t_{\text{main}}/t_{\text{cache}} \)

• if \( t_{\text{main}}/t_{\text{cache}} \) is 10 then a decrease of 1% in \( h \) [as \( h \to 1 \)] results in a 10% increase in \( t_{\text{eff}} \)
Temporal and Locality of Reference

• exploit the temporal locality and locality of reference inherent in *typical* programs

• high probability memory regions
  - recently executed code
  - recent stack accesses
  - recently accessed data

• if the memory references occur randomly, cache will have very little effect
• NB: see average on graph
A real example collected from an early IA32 PC running OSF/1
K-way Set Associative Cache with N Sets

N sets, K directories and L bytes per cache line
Searching a K-way Cache

- address mapped onto a particular set [set #]...
- by extracting bits from incoming address
- NB: tag, set # and offset

- consider an address that maps to set 1

- the set 1 tags of all K directories are compared with the incoming address tag simultaneously

- if a is match found [hit], corresponding data returned offset within cache line

- the K data lines in the set are accessed concurrently with the directory entries so that on a hit the data can be routed quickly to the output buffers

- if a match is NOT found [miss], read data from memory, place in cache line within set and update corresponding cache tag [choice of K positions]

- cache line replacement strategy [within a set] - Least Recently Used [LRU], pseudo LRU, random...
Caches

Searching a K-way Cache...

**TAGS**

<table>
<thead>
<tr>
<th>set 0</th>
<th>set 1</th>
<th>set 2</th>
<th>set 3</th>
<th>set N-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0326</td>
<td>1234*</td>
<td>9876</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DATA**

Directory 0

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
</tr>
</thead>
</table>

*each cache line holds 16 bytes organised as 4 x 32 bit words*

**Incoming address (32 bits or 8 nybbles)**

1234 001 8

tag set off

L=16, N=4096, K = K

NB: cache lines aligned on 16 byte boundaries
Cache Organisation

• the cache organisation is described using the following three parameters
  
  $L$: bytes per cache line [cache line or block size]
  $K$: cache lines per set [degree of associativity K-way]
  $N$: number of sets

• cache size $L \times K \times N$ bytes

• $N = 1$
  - fully associative cache, incoming address tag compared with ALL cache tags
  - address can map to any one of the $K$ cache lines

• $K = 1$
  - direct mapped cache, incoming address tag compared with ONLY ONE cache tag
  - address can be mapped ONLY onto a single cache line

• $N > 1$ and $K > 1$
  - set-associative [K-way cache]