Cache operation – overview

• CPU requests contents of memory location
• Check cache for this data
• If present, get from cache (fast)
• If not present, read required block from main memory to cache
• Then deliver from cache to CPU
• Cache includes tags to identify which block of main memory is in each cache slot
Cache Read Operation - Flowchart

START

Receive address RA from CPU

Is block containing RA in cache?

Yes

Fetch RA word and deliver to CPU

No

Access main memory for block containing RA

Allocate cache line for main memory block

Load main memory block into cache line

DONE

Deliver RA word to CPU
Typical Cache Organization
Where does the Cache sit?
## Comparison of Cache Sizes

<table>
<thead>
<tr>
<th>Processor</th>
<th>Type</th>
<th>Year of Introduction</th>
<th>L1 Cache</th>
<th>L2 Cache</th>
<th>L3 Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM 360/85</td>
<td>Mainframe</td>
<td>1968</td>
<td>16 to 32 kB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PDP-11/70</td>
<td>Minicomputer</td>
<td>1975</td>
<td>1 kB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>VAX 11/780</td>
<td>Minicomputer</td>
<td>1978</td>
<td>16 kB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>IBM 3033</td>
<td>Mainframe</td>
<td>1978</td>
<td>64 kB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>IBM 3090</td>
<td>Mainframe</td>
<td>1985</td>
<td>128 to 256 kB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Intel 80486</td>
<td>PC</td>
<td>1989</td>
<td>8 kB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Pentium</td>
<td>PC</td>
<td>1993</td>
<td>8 kB/8 kB</td>
<td>256 to 512 KB</td>
<td>—</td>
</tr>
<tr>
<td>PowerPC 601</td>
<td>PC</td>
<td>1993</td>
<td>32 kB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PowerPC 620</td>
<td>PC</td>
<td>1996</td>
<td>32 kB/32 kB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PowerPC G4</td>
<td>PC/server</td>
<td>1999</td>
<td>32 kB/32 kB</td>
<td>256 KB to 1 MB</td>
<td>2 MB</td>
</tr>
<tr>
<td>IBM S/390 G4</td>
<td>Mainframe</td>
<td>1997</td>
<td>32 kB</td>
<td>256 KB</td>
<td>2 MB</td>
</tr>
<tr>
<td>IBM S/390 G6</td>
<td>Mainframe</td>
<td>1999</td>
<td>256 kB</td>
<td>8 MB</td>
<td>—</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>PC/server</td>
<td>2000</td>
<td>8 kB/8 kB</td>
<td>256 KB</td>
<td>—</td>
</tr>
<tr>
<td>IBM SP</td>
<td>High-end server/ supercomputer</td>
<td>2000</td>
<td>64 kB/32 kB</td>
<td>8 MB</td>
<td>—</td>
</tr>
<tr>
<td>Itanium</td>
<td>PC/server</td>
<td>2001</td>
<td>16 kB/16 kB</td>
<td>96 KB</td>
<td>4 MB</td>
</tr>
<tr>
<td>SGI Origin 2001</td>
<td>High-end server</td>
<td>2001</td>
<td>32 kB/32 kB</td>
<td>4 MB</td>
<td>—</td>
</tr>
<tr>
<td>Itanium 2</td>
<td>PC/server</td>
<td>2002</td>
<td>32 kB</td>
<td>256 KB</td>
<td>6 MB</td>
</tr>
<tr>
<td>IBM POWER5</td>
<td>High-end server</td>
<td>2003</td>
<td>64 kB</td>
<td>1.9 MB</td>
<td>36 MB</td>
</tr>
<tr>
<td>CRAY XD-1</td>
<td>Supercomputer</td>
<td>2004</td>
<td>64 kB/64 kB</td>
<td>1 MB</td>
<td>—</td>
</tr>
<tr>
<td>IBM POWER6</td>
<td>PC/server</td>
<td>2007</td>
<td>64 kB/64 kB</td>
<td>4 MB</td>
<td>32 MB</td>
</tr>
<tr>
<td>IBM z10</td>
<td>Mainframe</td>
<td>2008</td>
<td>64 kB/128 kB</td>
<td>3 MB</td>
<td>24-48 MB</td>
</tr>
</tbody>
</table>
Mapping Memory to Cache

• Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.

• Further, a means is needed for determining which main memory block currently occupies a cache line.

• The choice of the mapping function dictates how the cache is organized.

• Three techniques can be used: direct, associative, and set associative.
Direct Mapping Example

Our example includes the following elements:

• The cache can hold 64 KBytes.

• Data are transferred between main memory and the cache in blocks of 4 bytes each.
  – This means that the cache is organized as $16K = 2^{14}$ lines of 4 bytes each.

• The main memory consists of 16 Mbytes, with each byte directly addressable by a 24-bit address ($2^{24} = 16M$).

• Thus, for mapping purposes, we can consider main memory to consist of 4M blocks of 4 bytes each.
Direct Mapping

Direct mapping, maps each block of main memory into only one possible cache line. The mapping is expressed as:

\[ i = j \mod m \]

Where:
- \( i \) = cache line number
- \( j \) = main memory block number
- \( m \) = number of lines in the cache
Direct Mapping from Cache to Main Memory

First $m$ blocks of main memory (equal to size of cache)

(a) Direct mapping

$b =$ length of block in bits
$t =$ length of tag in bits
## Direct Mapping

### Address Structure

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
  - 8 bit tag (=22-14)
  - 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

<table>
<thead>
<tr>
<th>Tag</th>
<th>Line or Slot</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-r</td>
<td>w</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>
The Figure shows the mapping for the first blocks of main memory. Each block of main memory maps into one unique line of the cache.

The next blocks of main memory map into the cache in the same fashion; that is, block $B_m$ of main memory maps into line $L_0$ of cache, block $B_{m+1}$ maps into line $L_1$, and so on.
The mapping function is easily implemented using the main memory address as illustrated:
• For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant $w$ bits identify a unique word or byte within a block of main memory.

• The remaining $s$ bits specify one of the $2^s$ blocks of main memory. The cache logic interprets these $s$ bits as a tag of $s-r$ bits (most significant portion) and a line field of $r$ bits. This latter field identifies one of the $m=2^r$ lines of the cache.
Address length = \((s+w)\) bits
Number of addressable units = \(2^{s+w}\) words/bytes
Block size = line size = \(2^w\) words/bytes
Number of blocks in main memory = \(\frac{2^{s+w}}{2^w} = 2^s\)
Number of lines in cache = \(m = 2^r\)
Size of cache = \(2^{r+w}\) words/bytes
Size of tag = \((s-r)\) bits
## Direct Mapping

### Cache Line Table

<table>
<thead>
<tr>
<th>Cache line</th>
<th>Main Memory blocks held</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0, m, 2m, 3m...2s-m</td>
</tr>
<tr>
<td>1</td>
<td>1, m+1, 2m+1...2s-m+1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>m-1</td>
<td>m-1, 2m-1, 3m-1...2s-1</td>
</tr>
</tbody>
</table>
Direct Mapping Cache Organization

Memory Address

Tag | Line | Word

s-r
r
w

Compare

s-w

(hit in cache)

1 if match
0 if no match

0 if match
1 if no match

Cache

Tag | Data

L_0

L_1

L_{m-1}

Main Memory

WO
W1
W2
W3

B_0

W_{4j}
W_{4j+1}
W_{4j+2}
W_{4j+3}

B_j
Direct Mapping Example

16-MByte main memory

Main memory address =

Note: Memory address values are in binary representation; other values are in hexadecimal.
• The direct mapping technique is simple and inexpensive to implement. Its main disadvantage is that there is a fixed cache location for any given block.

• Thus, if a program happens to reference words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the hit ratio will be low (a phenomenon known as *thrashing*).