Cache Memory

Internal Memory
- Processor Memory – Registers
- Control Unit Memory
- Cache Memory – L1, L2, L3
- Main Memory – Dynamic Random Access Memory (DRAM)

External Memory (Peripheral)
- Fixed hard Disk
- Removable Hard Disk
- Optical Disks – CD, DVD
- Solid State Memory – Flash Drives, Memory Cards
- Magnetic Tape

Unit of Transfer: number of electrical lines ↔ memory module

Word:
- Unit of memory organization, i.e., the “natural” amount of memory that is retrievable in a single cycle
- Number of bits used to represent an integer
- Equal to the instruction length
  - Intel x86 word size 32 bits – instruction lengths vary
  - Cray C90 64 bit word, instruction length 46 bits

Addressable Units
- Word
- Byte – address length A bits, yields the number of addressable units \( N = 2^A \)

Unit of Transfer
- Main Memory -- number of memory bits read/written during a single event, i.e., number of lines into and out of the memory module
- External Memory – blocks

Access Methods
- Sequential Access – magnetic tapes
  - sequential searching, counting, waiting
  - access time is highly variable

- Direct Access – disk drives
  - (track, sector) ↔ general vicinity
  - sequential searching, counting, waiting
  - access time is variable within strict limits

- Random Access – main memory
  - memory address ↔ data item
  - access time is
    - independent of previous searches
    - close to being constant

- Associative Access – cache memory
  - comparison of selected bit locations within a word, for a selected match, simultaneously
  - word is retrieved based on a portion of its contents rather than its address
  - access time is constant
Performance

- **Access Time (Latency)**
  - Random Access Memory
    - time differential: ((data stored or made available) – (address presented))
  - Non-Random Access Memory
    - time differential: ((read-write mechanism positioned at desired location) – (track, sector presented))

- **Memory Cycle Time – System Bus**
  - Access Time + time required for transient signals to die +
    time required to regenerate data after destructive reads

- **Transfer Time** – rate at which data is transferred to a memory unit
  - Random Access Memory
    - transfer time = 1/(cycle time)
  - Non-Random Access Memory
    - transfer time

  \[ T_N = T_A + \frac{n}{R} \]

  \( T_N \): average time to read/write N bits
  \( T_A \): average access time
  \( n \): number of bits
  \( R \): transfer rate measured in bits/second (bps)

- **Two Level Memory Access**

  \[
  \begin{array}{lll}
  \text{Level 1} & 1000 \text{ words} & \text{access time} \ 0.01 \mu\text{s} \\
  \text{Level 2} & 100,000 \text{ words} & \text{access time} \ 0.1 \mu\text{s} \\
  \end{array}
  \]

  \[ hit \ ratio = \frac{\text{Level 1 memory hits}}{\text{total memory hits}} \]

  95% memory accesses from Level 1
  5% memory accesses from Level 2

  \((0.95)(0.01 \mu\text{s}) + (0.05)(0.01 \mu\text{s} + 0.1 \mu\text{s}) \Rightarrow 0.0095 \mu\text{s} + 0.0055 \mu\text{s} \Rightarrow 0.015 \mu\text{s}\)

- **Locality of Reference**
  - Instructions
    - Loops, Subroutines
  - Data
    - Tables, Arrays, Clustered Data Sets

- **CPU** ↔ **L1 Cache** ↔ **L2 Cache** ↔ **Memory Register Set (Cache)** ↔ **Main Memory**

- **External Storage** ↔ **Buffer Memory or Disk Cache**

  IBM Mainframes
  Expanded Storage

  Disk Read/Write
  Block Clusters
### Cache Memory

The diagram depicts the flow of data between the **CPU**, **L1 Cache**, **L2 Cache**, and **Main Memory**. The flow is indicated by arrows, showing the direction of data movement. The **CPU** accesses the **L1 Cache** (faster) before accessing the **L2 Cache** and **Main Memory** (slower). This structure optimizes data access by placing frequently accessed data in the caches, reducing the latency of memory access.

### Locality of Reference

Locality of reference is a principle in computer science that states that a program's instructions and data are referenced in a local manner. This implies that if a program references a particular piece of data, it is likely to access nearby data as well. This locality can be exploited by caching mechanisms to improve performance.

### Blocks of Memory

The image illustrates how memory is divided into blocks of size **K** words. Each block is contained within a **Line**, and **M** such lines constitute the **Main Memory**. The **Tag** bits identify the block currently stored in a line, and control bits, such as modified bit, read bit, etc., are used for additional functionality.

### Line Size and Block Length

The **Line Size** is equal to the **Block Length**, i.e., **K Words**. This ensures that data is accessed in cohesive chunks, aligning with the locality of reference principle.

### Byte Addressable Machines

Byte addressable machines can have **Lines** as small as 32 bits. The number of **m** bits is much smaller than the word length, i.e., **m << M**.

### Summary

- **Main Memory** contains **2^n addressable words**.
- Each word has a unique **n-bit address**.
- **M** blocks each consisting of **K words**.
- The number of blocks is given by **M = 2^n / K**.
• Logical Cache, i.e., Virtual Cache
  Faster response time within each process
  Context switch ➔ flush cache memory
  OR
  add additional bits to each line which identifies the process that is using that line

• Physical Cache
  Slower response time since each cache access has to also invoke the MMU
  Context switch does not require that the cache memory be flushed

• Cache Size Parameters
  Small ➔ cost per bit of cache ≅ cost per bit of main memory
  Large ➔ overall average access time of (main + cache) memory ≅ access time of cache memory
  \( \Delta \) cache size ➔ \( \Delta \) logic gates ➔ \( \Delta \) cache access time
  chip & board size limits cache memory size

• Mapping Function
  Main Memory Blocks ➔ Cache Memory Lines
  Determining which memory block occupies a specific cache line

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>Main Memory Blocks Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0, m, 2m, ..., ( 2^{s} - m )</td>
</tr>
<tr>
<td>1</td>
<td>1, m + 1, 2m + 1, ..., ( 2^{s} - m + 1 )</td>
</tr>
<tr>
<td>2</td>
<td>2, m + 2, 2m + 2, ..., ( 2^{s} - m + 2 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>m - 1</td>
<td>m - 1, 2m - 1, 3m - 1, ..., ( 2^{s} - 1 )</td>
</tr>
</tbody>
</table>

\[ i = j \mod m \] where \( i \) : cache line number, \( j \) : main memory block number, \( m \) : number of cache lines

<table>
<thead>
<tr>
<th>Cache Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m = 16K = 2^{14} )</td>
</tr>
<tr>
<td>( i = j \mod 2^{14} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>Block Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000000, 010000, ..., FF0000</td>
</tr>
<tr>
<td>1</td>
<td>000004, 010004, ..., FF0004</td>
</tr>
<tr>
<td>2^{14} - 1</td>
<td>00FFFC, 01FFFC, ... , FFFFC</td>
</tr>
</tbody>
</table>

Assumptions
- cache holds 64 KB
- data transfer block size == 4 bytes
- cache line size == 4 bytes
- number of cache lines == 16K, i.e., \( 2^{14} \) lines of 4 bytes each
- main memory == 16 MB; byte addressable; 24-bit address (\( 2^{24} = 16 \) MB)
- main memory == 4 MB blocks (4 bytes/block)

• Direct Mapping

If two different blocks have the same tag number they map onto different lines, e.g.,
Block 000101100110011100110011 containing FEDCBA98 maps to line 0CE7 with tag 16
Block 000101101111111111 containing 12345678 maps to line 3FFF with tag 16

Implementation is inexpensive
If two different blocks map to the same line, repeated references to these two blocks will result in continual swapping of the blocks, i.e., thrashing!
Save swapped out line in a victim cache
Victim Cache – fully associative cache (4-16 cache lines) residing between direct mapped L1 cache & next level of memory
• Associative Mapping
  o each main memory block may be loaded into any line of the cache
  o memory address $\leftrightarrow$ Tag + Word
    - Tag is used to identify a specific memory block
    - Word $=\text{offset within the line}$ used to identify a specific word
  o memory address does not identify the line
  o number of lines is not determined by the address format
  o control logic simultaneously examines every line's Tag for a match to the memory address's Tag

Disadvantage – complex circuitry required to implement associative memory retrieval
Advantage – flexibility as to where a new block may be stored in the cache

• Set Associative Mapping
  o Set $=\text{Group of Lines}$
  o Cache $=\text{Group of Sets}$
  o $k$-way set-associative mapping function
    $$m = v \times k$$
    $$i = j \mod v$$
    where
    - $i$ : cache set number
    - $j$ : main memory block number
    - $m$ : number of lines in the cache
    - $v$ : number of sets
    - $k$ : number of lines in each set

Block $B_j$ can be mapped into any of the lines of the set $j$

$$v == m \& k == 1 \Rightarrow \text{direct mapping}$$
$$v == 1 \& k == m \Rightarrow \text{associative mapping}$$
$$v == m/2 \& k == 2 \Rightarrow \text{2-way set-associative cache memory}$$
  Significantly improves hit ratio over direct mapping
$$v == m/4 \& k == 4 \Rightarrow \text{4-way set-associative cache memory}$$
  Modest improvement over 2-way for small additional cost

• Physical implementation of a set-associative cache
  $\checkmark$ $v$ number of individual associative caches
  $\checkmark$ $k$ number of individual direct associative caches
  $\Rightarrow$ each direct associative cache $\leftrightarrow$ way which consists of $v$ lines
  $\Rightarrow$ first $v$ blocks of main memory are directly mapped to the $v$ lines of each way

Data retrieval – data item
  - main memory address 160000
  - 0001 0110 0000 0000 0000 00 00
  - data content 7777 7777
  - yields set number 0000
  - potential tags 000 or 02C
  - associative matching of tags yields 02C
  - returning content 7777 7777
Replacement Algorithms

- **Direct Mapping** -- ∀ block ∃! line, i.e., for every block there exists only one unique line, i.e., there is no choice

- **Associative & Set Associative**
  - LRU, i.e., Least Recently Used – select block with longest time in cache with no memory references
  - 2-way set-associative: each line has a USE bit – when a line is referenced its USE bit is set to 1 and the USE bit of the other line is set to 0

  ![Diagram of 2-way set-associative cache](image)

  - replacement choice for the set will be the line with the USE bit == 0

- **n-way associative & associative**: maintain a separate list of line numbers when a line is referenced, its line number is placed at the front of the list replacement candidates are taken from the back of the list

- **FIFO**, i.e., first-in, first-out – select block with longest time in cache
  - round-robin -- circular buffer

- **LRU**, i.e., least recently used – select block with fewest references
  - each line has a counter to record the number of references

  ![Counter diagram](image)

- **Random choice** – select block using a random number generator

Write Policy

- old block in cache has not been altered → overwrite old block with new block
- old block in cache has been altered → write old block to main memory before overwriting with the new block

- **Write-Through** – all write operations change both the cache and the main memory any other processor-cache module must monitor main memory traffic to maintain consistency

  traffic bottleneck

- **Write-Back** – updates are written only to the cache
  - when line is modified it’s dirty bit is set
  - when the line is selected for replacement, the line needs to be written to main memory only if it’s dirty bit is set

  portions of main memory are invalid

  access of I/O modules must be through the cache

  complex circuitry & potential bottleneck

<table>
<thead>
<tr>
<th>Percentage of Memory Reference Writes, i.e.,</th>
<th>write memory references</th>
<th>total memory references</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal computation 15%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vector-vector multiplication 33%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix transposition 50%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bus Organization
multiple devices may each have a cache memory module
even with write-through, multiple caches may hold invalid data

- Bus Watching with Write-Through
  - each cache controller monitors write operations to main memory by other bus masters
  - if a write is to a location in main memory that also resides in the local cache,
    the local cache data items are invalidated

- Hardware Transparency
  - All cache updates are written to main memory;
    any matching words in other caches are also updated

- Non-cacheable Memory
  - Shared memory
    - Is that portion of memory shared by more than one processor
    - declared to be non-cacheable, i.e., it is never copied into cache memory
    - all references to shared memory are cache misses

Line Size
Principle of Locality

- Normal Computation  8 Bytes ≤ Line Size ≤ 64 Bytes
- High Performance  64 Bytes ≤ Line Size ≤ 128 bytes

Multilevel Caches
- L1 : on-chip cache
  - bus access is eliminated
  - shorter data paths

- L2 : external cache, i.e., off-chip cache
  - processor does not need to engage in bus access to DRAM nor ROM
  - If L2 is implemented as a SRAM, i.e., static RAM chip, which matches the bus speed
    then data can be accessed using a zero-wait state transaction
  - separate data path may be used to reduce traffic on the data bus

- L2 may be placed on-chip yielding faster response times
  - L3 off-chip
  - L3 on-chip

Split Caches – parallel instruction execution, prefetching predicted future instructions
Data Caches
Instruction Caches

Unified Cache – higher hit ratio –
Execution pattern
- More instruction fetches than data fetches ➔ cache fills with instructions
- More data fetches than instruction fetches ➔ cache fills with data
Performance Characteristics of Two-Level Memories

Spatial Locality – clustered memory locations
- Sequential Instructions
- Sequential Data Locations
- Using larger cache blocks
- Incorporating prefetching mechanisms into the cache memory logic

Temporal Locality – repeatedly access a set of memory locations
- Loops
- Maintaining recently used data and instructions in cache memory