Chapter 4
Cache Memory

minor modifications by N. Santiago
Characteristics

- Location
- Capacity
- Unit of transfer
- Access method
- Performance
- Physical type
- Physical characteristics
- Organisation
Location

- CPU
- Internal
- External
**Capacity**

- **Word size**
  - The natural unit of organisation
- **Number of words**
  - or Bytes
Unit of Transfer

- **Internal**
  - Usually governed by data bus width

- **External**
  - Usually a block which is much larger than a word

- **Addressable unit**
  - Smallest location which can be uniquely addressed
    - Word internally
    - Cluster on M$ disks
Access Methods (1)

• Sequential
  — Start at the beginning and read through in order
  — Access time depends on location of data and previous location
  — e.g. tape

• Direct
  — Individual blocks have unique address
  — Access is by jumping to vicinity plus sequential search
  — Access time depends on location and previous location
  — e.g. disk
Access Methods (2)

- **Random**
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM

- **Associative**
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of location or previous access
  - e.g. cache
Memory Hierarchy

- Registers
  - In CPU

- Internal or Main memory
  - May include one or more levels of cache
  - “RAM”

- External memory
  - Backing store
Memory Hierarchy - Diagram

- Inboard Memory
- Registers
- Offboard Storage
- Magnetic Disk
- Main Memory
- CD-ROM
- DVD+RW
- Magnetic Tape
- CD-RW
- DVD-RAM
- MO
- WORM
Performance

- Access time
  - Time between presenting the address and getting the valid data

- Memory Cycle time
  - Time may be required for the memory to "recover" before next access
  - Cycle time is access + recovery

- Transfer Rate
  - Rate at which data can be moved
Physical Types

- Semiconductor
  - RAM
- Magnetic
  - Disk & Tape
- Optical
  - CD & DVD
Physical Characteristics

- Decay
- Volatility
- Erasable
- Power consumption
Organisation

- Physical arrangement of bits into words
- Not always obvious
- e.g. interleaved
The Bottom Line

- How much?
  - Capacity
- How fast?
  - Time is money
- How expensive?
Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape
So you want fast?

- It is possible to build a computer which uses only static RAM (see later)
- This would be very fast
- This would need no cache
  - How can you cache cache?
- This would cost a very large amount
Locality of Reference

- During the course of the execution of a program, memory references tend to cluster
- e.g. loops
Locality

- Principle of Locality:
  - Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
  - **Temporal locality:** Recently referenced items are likely to be referenced in the near future.
  - **Spatial locality:** Items with nearby addresses tend to be referenced close together in time.

Locality Example:

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference sum each iteration: **Temporal locality**

- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

```c
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```
Cache

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module
Cache and Main Memory

(a) Single cache

(b) Three-level cache organization
Cache/Main Memory Structure

(a) Cache

(b) Main memory
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

1. **START**
2. Receive address RA from CPU
3. Is block containing RA in cache?
   - Yes: Fetch RA word and deliver to CPU
   - No: Access main memory for block containing RA
4. Allocate cache line for main memory block
5. Load main memory block into cache line
6. Deliver RA word to CPU
7. **DONE**
Cache Design

- Addressing
- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches
Cache Addressing

- Where does cache sit?
  - Between processor and virtual memory management unit
  - Between MMU and main memory
- Logical cache (virtual cache) stores data using virtual addresses
  - Processor accesses cache directly, not through physical cache
  - Cache access faster, before MMU address translation
  - Virtual addresses use same address space for different applications
    - Must flush cache on each context switch
- Physical cache stores data using main memory physical addresses
Size does matter

• Cost
  — More cache is expensive

• Speed
  — More cache is faster (up to a point)
  — Checking cache for data takes time
Typical Cache Organization
# 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>CRAY MTA</td>
<td>Supercomputer</td>
<td>2000</td>
<td>8 KB</td>
<td>2 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>
</tbody>
</table>
Mapping Function

- Cache of 64kB
- Cache block of 4 bytes
  - i.e. cache is 16k ($2^{14}$) lines of 4 bytes
- 16MB main memory
- 24 bit address
  - ($2^{24} = 16M$)
Direct Mapping

- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Address is in two parts
- Least Significant w bits identify unique word
- Most Significant s bits specify one memory block
- The MSBs are split into a cache line field r and a tag of s-r (most significant)
**Direct Mapping Address Structure**

<table>
<thead>
<tr>
<th>Tag s-r</th>
<th>Line or Slot r</th>
<th>Word w</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>14</td>
<td>2</td>
</tr>
</tbody>
</table>

- 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
Direct Mapping from Cache to Main Memory

(a) Direct mapping

- b = length of block in bits
- t = length of tag in bits

First m blocks of main memory (equal to size of cache)
## 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
Direct Mapping Example

Main memory address (binary)

00 00 00000000000000000000000000000000 13579246

00 00 00000000011111111111111111111111 11235813

16 16 00010110000000000000000000000000 16 16 11223344 12345678

16 16 00010110001100110011001100110011 16 16 11223344 12345678

16 16 00010110111111111111111111111111 16 16 11223344 12345678

FF FF 11111111110000000000000000000000 111111111111111111111111 12345678

Note: Memory address values are in binary representation; other values are in hexadecimal.

16-MByte main memory

Main memory address =

8 bits 14 bits 2 bits
Direct Mapping Summary

- Address length = \((s + w)\) bits
- Number of addressable units = \(2^{s+w}\) words or bytes
- Block size = line size = \(2^w\) words or bytes
- Number of blocks in main memory = \(2^{s+w/2w} = 2^s\)
- Number of lines in cache = \(m = 2^r\)
- Size of tag = \((s - r)\) bits
Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high
Victim Cache

- Lower miss penalty
- Remember what was discarded
  - Already fetched
  - Use again with little penalty
- Fully associative
- 4 to 16 cache lines
- Between direct mapped L1 cache and next memory level
Associative Mapping

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line’s tag is examined for a match
- Cache searching gets expensive
Associative Mapping from Cache to Main Memory

one block of main memory

cache memory
Fully Associative Cache Organization

Memory Address

Tag

Word

Cache

Tag

Data

L_0

L_j

L_{m-1}

Main Memory

W_0

W_1

W_2

W_3

B_0

W_{4j}

W_{4j+1}

W_{4j+2}

W_{4j+3}

B_j

Compare

1 if match
0 if no match

(hit in cache)

0 if match
1 if no match

(miss in cache)
Associative Mapping Example

Main Memory Address (binary)

Tag (hex)
000000
000001

Tag

Word

Data

13579246

058CE6
058CE7
058CE8

FEDCBA98

3FFFED
3FFFFF
3FFFFF

3FFFD
000000
3FFFFF

22 bits
32 bits
16 Kline Cache

33333333
11223344
24682468

22 bits
2 bits

Note: Memory address values are in binary representation; other values are in hexadecimal

16 MByte Main Memory
**Associative Mapping Address Structure**

<table>
<thead>
<tr>
<th>Tag 22 bit</th>
<th>Word 2 bit</th>
</tr>
</thead>
</table>

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block
- e.g.
  - Address: FFFFFFFC
  - Tag:  FFFFFFFC
  - Data:  24682468
  - Cache line: 3FFF
Associative Mapping Summary

- Address length = \((s + w)\) bits
- Number of addressable units = \(2^{s+w}\) words or bytes
- Block size = line size = \(2^w\) words or bytes
- Number of blocks in main memory = \(2^{s+w}/2^w = 2^s\)
- Number of lines in cache = undetermined
- Size of tag = \(s\) bits
Set Associative Mapping

- Cache is divided into a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set
Set Associative Mapping Example

- 13 bit set number
- Block number in main memory is modulo $2^{13}$
- 000000, 00A000, 00B000, 00C000 ... map to same set
Mapping From Main Memory to Cache: v Associative

First v blocks of main memory (equal to number of sets)
Mapping From Main Memory to Cache: k-way Associative

First $v$ blocks of main memory (equal to number of sets)
K-Way Set Associative Cache Organization

Memory Address

Tag  Set  Word
s-d  d    w

Compare

1 if match
0 if no match

(hit in cache)

Cache

Tag  Data

F_0
F_1
\ldots
F_{k-1}
F_k
F_{k+1}
F_{2k-1}

Set 0
Set 1

Main Memory

B_0
B_1
\ldots
B_j
\ldots

0 if match
1 if no match

(miss in cache)
Set Associative Mapping Address Structure

<table>
<thead>
<tr>
<th>Tag 9 bit</th>
<th>Set 13 bit</th>
<th>Word 2 bit</th>
</tr>
</thead>
</table>

- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit
- e.g

<table>
<thead>
<tr>
<th>Address number</th>
<th>Tag</th>
<th>Data</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1FF 7FFC</td>
<td>1FF</td>
<td>12345678</td>
<td>1FFF</td>
</tr>
<tr>
<td>001 7FFC</td>
<td>001</td>
<td>11223344</td>
<td>1FFF</td>
</tr>
</tbody>
</table>
Two Way Set Associative Mapping Example

Note: Memory address values are in binary representation; other values are in hexadecimal.
Set Associative Mapping Summary

- Address length = \( (s + w) \) bits
- Number of addressable units = \( 2^{s+w} \) words or bytes
- Block size = line size = \( 2^w \) words or bytes
- Number of blocks in main memory = \( 2^d \)
- Number of lines in set = \( k \)
- Number of sets = \( v = 2^d \)
- Number of lines in cache = \( kv = k \times 2^d \)
- Size of tag = \( (s - d) \) bits
Direct and Set Associative Cache Performance Differences

- Significant up to at least 64kB for 2-way
- Difference between 2-way and 4-way at 4kB much less than 4kB to 8kB
- Cache complexity increases with associativity
- Not justified against increasing cache to 8kB or 16kB
- Above 32kB gives no improvement
- (simulation results)
Figure 4.16
Varying Associativity over Cache Size

- Hit ratio vs. Cache size (bytes)
- Cache sizes: 1k, 2k, 4k, 8k, 16k, 32k, 64k, 128k, 256k, 512k, 1M
- Associativity levels: direct, 2-way, 4-way, 8-way, 16-way

The graph illustrates the varying associativity over cache size, showing how hit ratios change with different cache sizes and associativity configurations.
Replacement Algorithms (1)
Direct mapping

- No choice
- Each block only maps to one line
- Replace that line
Replacement Algorithms (2)
Associative & Set Associative

- Hardware implemented algorithm (speed)
- Least Recently used (LRU)
  - e.g. in 2 way set associative
    - Which of the 2 block is lru?
- First in first out (FIFO)
  - replace block that has been in cache longest
- Least frequently used
  - replace block which has had fewest hits
- Random
Write Policy

- Must not overwrite a cache block unless main memory is up to date
- Multiple CPUs may have individual caches
- I/O may address main memory directly
**Write through**

- All writes go to main memory as well as cache
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic
- Slows down writes

- Remember bogus write through caches!
Write back

- Updates initially made in cache only
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- N.B. 15% of memory references are writes
Line Size

- Retrieve not only desired word but a number of adjacent words as well
- Increased block size will increase hit ratio at first
  - the principle of locality
- Hit ratio will decreases as block becomes even bigger
  - Probability of using newly fetched information becomes less than probability of reusing replaced
- Larger blocks
  - Reduce number of blocks that fit in cache
  - Data overwritten shortly after being fetched
  - Each additional word is less local so less likely to be needed
- No definitive optimum value has been found
- 8 to 64 bytes seems reasonable
- For HPC systems, 64- and 128-byte most common
Multilevel Caches

- High logic density enables caches on chip
  - Faster than bus access
  - Frees bus for other transfers
- Common to use both on and off chip cache
  - L1 on chip, L2 off chip in static RAM
  - L2 access much faster than DRAM or ROM
  - L2 often uses separate data path
  - L2 may now be on chip
  - Resulting in L3 cache
    - Bus access or now on chip...
Hit Ratio (L1 & L2)
For 8 kbytes and 16 kbyte L1

Hit ratio

L2 Cache size (bytes)

L1 = 16k

L1 = 8k
Unified v Split Caches

- One cache for data and instructions or two, one for data and one for instructions

Advantages of unified cache
- Higher hit rate
  - Balances load of instruction and data fetch
  - Only one cache to design & implement

Advantages of split cache
- Eliminates cache contention between instruction fetch/decode unit and execution unit
  - Important in pipelining
Pentium 4 Cache

- 80386 – no on chip cache
- 80486 – 8k using 16 byte lines and four way set associative organization
- Pentium (all versions) – two on chip L1 caches
  - Data & instructions
- Pentium III – L3 cache added off chip
- Pentium 4
  - L1 caches
    - 8k bytes
    - 64 byte lines
    - four way set associative
  - L2 cache
    - Feeding both L1 caches
    - 256k
    - 128 byte lines
    - 8 way set associative
  - L3 cache on chip
## Intel Cache Evolution

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
<th>Processor on which feature first appears</th>
</tr>
</thead>
<tbody>
<tr>
<td>External memory slower than the system bus.</td>
<td>Add external cache using faster memory technology.</td>
<td>386</td>
</tr>
<tr>
<td>Increased processor speed results in external bus becoming a bottleneck for cache access.</td>
<td>Move external cache on-chip, operating at the same speed as the processor.</td>
<td>486</td>
</tr>
<tr>
<td>Internal cache is rather small, due to limited space on chip</td>
<td>Add external L2 cache using faster technology than main memory</td>
<td>486</td>
</tr>
<tr>
<td>Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit’s data access takes place.</td>
<td>Create separate data and instruction caches.</td>
<td>Pentium</td>
</tr>
<tr>
<td>Increased processor speed results in external bus becoming a bottleneck for L2 cache access.</td>
<td>Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.</td>
<td>Pentium Pro</td>
</tr>
<tr>
<td>Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.</td>
<td>Move L2 cache on to the processor chip.</td>
<td>Pentium II</td>
</tr>
<tr>
<td></td>
<td>Add external L3 cache.</td>
<td>Pentium III</td>
</tr>
<tr>
<td></td>
<td>Move L3 cache on-chip.</td>
<td>Pentium 4</td>
</tr>
</tbody>
</table>
Pentium 4 Block Diagram

- Out-of-order execution logic
- L1 instruction cache (12K μops)
- Instruction fetch/decode unit
- System Bus
- L3 cache (1 MB)
- 64 bits
- L2 cache (512 KB)
- 256 bits
- Integer register file
- FP register file
- Load address unit
- Store address unit
- Simple integer ALU
- Simple integer ALU
- Complex integer ALU
- FP/ MMX unit
- FP move unit
Pentium 4 Core Processor

• Fetch/Decode Unit
  — Fetches instructions from L2 cache
  — Decode into micro-ops
  — Store micro-ops in L1 cache

• Out of order execution logic
  — Schedules micro-ops
  — Based on data dependence and resources
  — May speculatively execute

• Execution units
  — Execute micro-ops
  — Data from L1 cache
  — Results in registers

• Memory subsystem
  — L2 cache and systems bus
Pentium 4 Design Reasoning

- Decodes instructions into RISC like micro-ops before L1 cache
- Micro-ops fixed length
  - Superscalar pipelining and scheduling
- Pentium instructions long & complex
- Performance improved by separating decoding from scheduling & pipelining
  - (More later – ch14)
- Data cache is write back
  - Can be configured to write through
- L1 cache controlled by 2 bits in register
  - CD = cache disable
  - NW = not write through
  - 2 instructions to invalidate (flush) cache and write back then invalidate
- L2 and L3 8-way set-associative
  - Line size 128 bytes
# ARM Cache Features

<table>
<thead>
<tr>
<th>Core</th>
<th>Cache Type</th>
<th>Cache Size (kB)</th>
<th>Cache Line Size (words)</th>
<th>Associativity</th>
<th>Location</th>
<th>Write Buffer Size (words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM720T</td>
<td>Unified</td>
<td>8</td>
<td>4</td>
<td>4-way</td>
<td>Logical</td>
<td>8</td>
</tr>
<tr>
<td>ARM920T</td>
<td>Split</td>
<td>16/16 D/I</td>
<td>8</td>
<td>64-way</td>
<td>Logical</td>
<td>16</td>
</tr>
<tr>
<td>ARM926EJ-S</td>
<td>Split</td>
<td>4-128/4-128 D/I</td>
<td>8</td>
<td>4-way</td>
<td>Logical</td>
<td>16</td>
</tr>
<tr>
<td>ARM1022E</td>
<td>Split</td>
<td>16/16 D/I</td>
<td>8</td>
<td>64-way</td>
<td>Logical</td>
<td>16</td>
</tr>
<tr>
<td>ARM1026EJ-S</td>
<td>Split</td>
<td>4-128/4-128 D/I</td>
<td>8</td>
<td>4-way</td>
<td>Logical</td>
<td>8</td>
</tr>
<tr>
<td>Intel StrongARM</td>
<td>Split</td>
<td>16/16 D/I</td>
<td>4</td>
<td>32-way</td>
<td>Logical</td>
<td>32</td>
</tr>
<tr>
<td>Intel Xscale</td>
<td>Split</td>
<td>32/32 D/I</td>
<td>8</td>
<td>32-way</td>
<td>Logical</td>
<td>32</td>
</tr>
<tr>
<td>ARM1136-JF-S</td>
<td>Split</td>
<td>4-64/4-64 D/I</td>
<td>8</td>
<td>4-way</td>
<td>Physical</td>
<td>32</td>
</tr>
</tbody>
</table>
ARM Cache Organization

- Small FIFO write buffer
  - Enhances memory write performance
  - Between cache and main memory
  - Small c.f. cache
  - Data put in write buffer at processor clock speed
  - Processor continues execution
  - External write in parallel until empty
  - If buffer full, processor stalls
  - Data in write buffer not available until written
    - So keep buffer small
ARM Cache and Write Buffer Organization

Diagram showing the interaction between the processor, cache, write buffer, and main memory. The diagram illustrates the flow of data and access times between these components.
Internet Sources

- Manufacturer sites
  - Intel
  - ARM
- Search on cache