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
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
- Others
  - Bubble
  - Hologram
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
Cache

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

START

Receive address RA from CPU

Is block containing RA in cache?

No

Access main memory for block containing RA

Yes

Fetch RA word and deliver to CPU

Allocate cache line for main memory block

Load main memory block into cache line

DONE

Deliver RA word to CPU
Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches
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 MTAb</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 64kByte
- Cache block of 4 bytes
  - i.e. cache is 16k \(2^{14}\) lines of 4 bytes
- 16MBytes 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

#### 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>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 =

<table>
<thead>
<tr>
<th>Tag</th>
<th>Line</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>14</td>
<td>2</td>
</tr>
</tbody>
</table>
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
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 Address Structure

<table>
<thead>
<tr>
<th>Tag</th>
<th>22 bit</th>
<th>Word</th>
<th>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 | Tag | Data | Cache line
  - FFFFFFFC | FFFFFFFC24682468 | 3FFF
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
Two Way Set Associative Cache Organization
Set Associative Mapping
Address Structure

- 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>Tag 9 bit</th>
<th>Set 13 bit</th>
<th>Word 2 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address number</td>
<td>Tag</td>
<td>Data</td>
</tr>
<tr>
<td>1FF 7FFC</td>
<td>1FF</td>
<td>12345678</td>
</tr>
<tr>
<td>001 7FFC</td>
<td>001</td>
<td>11223344</td>
</tr>
</tbody>
</table>
Two Way Set
Associative Mapping Example

16 MByte Main Memory

Main Memory Address =

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