**Key Characteristics of Memories / Storage**

<table>
<thead>
<tr>
<th>Location</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor (main)</td>
<td>Access time</td>
</tr>
<tr>
<td>Internal (main)</td>
<td>Cycle time</td>
</tr>
<tr>
<td>External (secondary)</td>
<td>Transfer rate</td>
</tr>
</tbody>
</table>

**Capacity**
- Word size
- Number of words
- Unit of transfer
  - Word
  - Block

**Physical Characteristics**
- Semiconductor
- Magnetic
- Optical
- Magneto-Optical

**Access Method**
- Volatile/Non-volatile
- Erasable/Erasable

**Organization**
- Memory Hierarchy

---

**Goals**
- I want my memory lightning fast
- I want my memory to be gigantic in size
- Register access viewpoint
  - data access as fast as HW register
  - data size as large as memory
- Memory access viewpoint
  - data access as fast as memory
  - data size as large as disk

**Memory Hierarchy**
- Most often needed data kept close
- Access to small data sets can be made fast
  - simpler circuits
  - smaller gate delays
- Faster – more expensive
- Large can be bigger and cheaper (per byte)

**Principle of locality**
- In any given time period, memory references occur only to a small subset of the whole address space
- Temporal locality (ajallinen)
  - It is likely that a data item referenced a short time ago will be referenced again soon
- Spatial locality (alueellinen)
  - It is likely that a data item close to the one referenced a short time ago will be referenced soon

**Prob (small data set) = 95%**
- “Cost” (small data set) = 0.01 μs
- “Cost” (the rest) = 0.1 μs
- Aver. cost = 95% * 0.01 μs + 5% * 0.1 μs = 0.015 μs

**MEM:**
- Prob (small data set) = 95%
- Prob (the rest) = 5%
- “Cost” (small data set) = 0.01 μs
- “Cost” (the rest) = 0.1 μs
- Aver. cost = 95% * 0.01 μs + 5% * 0.1 μs = 0.015 μs

**Principle of locality (paikallisus)**
- In any given time period, memory references occur only to a small subset of the whole address space
- The reason why memory hierarchies work

**Memory Hierarchy**
- Most often needed data kept close
- Access to small data sets can be made fast
  - simpler circuits
  - smaller gate delays
- Faster – more expensive
- Large can be bigger and cheaper (per byte)

**Principle of locality**
- In any given time period
  - Memory references occur only to a small subset of the whole address space
  - Temporal locality (ajallinen)
    - It is likely that a data item referenced a short time ago will be referenced again soon
  - Spatial locality (alueellinen)
    - It is likely that a data item close to the one referenced a short time ago will be referenced soon
Cache Memory (välimuisti)

- How to access main memory as fast as registers?
- Locality → Use (CPU) cache!
  - Keep most probably referenced data in fast cache close to processor, and rest in memory
  - Most of data accesses only to cache
    - hit ratio 0.9-0.99
  - Cache is much smaller than main memory
  - Cache is (much) more expensive (per byte) than memory
- Note: file cache is another thing...

Cache Organization

Cache Read

Teemu’s Cheesecake

Register, on-chip cache, memory, disk, and tape speeds relative to times locating cheese for the cheese cake you are baking...

Cache Table

1 sec (cache) 10 sec (memory) 12 days (disk)

Radiator

Hand

Table

Moon

Europa (Jupiter)

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)

0.5 sec (register)

Hand

Refridgerator

0.5 sec (register)
Lecture 3: Memory, Cache

5.11.2010

Comp. Org II, Autumn 2010 3

Cache Design

<table>
<thead>
<tr>
<th>Cache Size &amp; Line Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many blocks help for temporal locality</td>
</tr>
<tr>
<td>Large blocks help for spatial locality</td>
</tr>
<tr>
<td>Larger cache is slower</td>
</tr>
<tr>
<td>Multi-level cache</td>
</tr>
</tbody>
</table>

Typical sizes:
- L1: 8 KB - 64 KB
- L2: 256 KB - 8 MB
- L3: 2 MB - 48 MB

Mapping

Which block (if any) contains the referenced memory location?
- Is the block in cache?
- Where in the cache is it located?

Solutions
- Direct mapping (suora kuvaus)
  - One possible location
- Fully associative mapping (yksikäsin asosiatiivinen)
  - Any possible location
- Set associative mapping (joukkosaasosiatiivinen)
  - Some possible locations

Typical sizes:
- L1: 8 KB - 64 KB
- L2: 256 KB - 8 MB
- L3: 2 MB - 48 MB

Cache simulation tools:
http://www.ecs.umass.edu/ece/koren/architecture/CACHE/frame0.htm

Direct Mapping

Each block has only one possible location (line) in cache
- determined by index bits
Several blocks may map into same cache line
- identified with different tag bits

Direct Mapping Example

Cache line size = Block size = 2^5 = 32 B

Fully Associative Mapping

Each block can be in any cache line
- tag must be complete block number

Fully Associative Mapping Example

Block number (in memory) Offset from the beginning of the block (in bytes)
34 bit address (byte address)

Cache size can be any number of blocks
Each block can be anywhere

Unique bits that are different for each block
Stored into cache line

Block size = 2^5 = 32 B
Fully Associative Example

```
ReadW I2, 0xB4
```

```
tag offset
00: 10110 12 34 56 78 9A 01 23 45
01: 10110 87 00 32 89 65 A1 B2 00
10: 10110 87 54 00 89 65 A1 B2 00
11: 10110 54 A7 00 91 23 66 32 11
```

```
```

```
block
```

```
Match
```

Fully Associative Mapping

- Lots of circuits
  - tag fields are long - wasted space?
  - each cache line tag must be compared in parallel with the memory address tag
    - lots of wires, comparison circuits
    - large surface area on chip
- Final comparison “or” has large gate delay
  - did any of these 64 comparisons match?
  - log2(64) = 6 levels of binary OR-gates
  - how about 262144 comparisons?
    - 18 levels?
  - Can use it only for small caches

Set Associative Mapping

- With set size \( k=2 \), each cache entry contain 2 blocks
- Use set (set index) field to find the cache entry
- Use tag to determine if the block belongs to the set
- Use offset to find the proper byte in the block

```
34 bit address (byte address)
tag set offset
```

```
Block size = \( 2^3 = 32 \) B
```

```
Unique bits that are different for each block, stored with block.
```

```
Nf of sets = \( v = 2^2 = 128 \text{ blocks} = 4 \text{ KB} \)
```

```
Total cache size = \( k*v = 2*4 \text{ KB} = 8 \text{ KB} \)
```

```
(16 blocks = 4 sets)
```

```
(8 blocks = 2 sets)
```

```
2-way Set Associative Cache
```

```
k=2 \rightarrow \text{two blocks in each set (~ in one cache entry) }
```

```
2 \text{ words in a block} = 8 \text{ bytes} \rightarrow 3 \text{ bits for byte offset}
```

```
Cache size \( 16 \text{ words} = 4 \text{ sets} \rightarrow 2 \text{ bits for set index}
```

```
Remaining 3 bits for tag
```

```
\text{Whole cache is one set!}
```

```
Each cache line is a set!
```

Set Associative Mapping

- Set associative cache with set size \( k=2 \)
  - 2-way cache (common)
- Degree of associativity = \( v \) = \( \text{nr of blocks in a set} \)
  - Large degree of associativity?
    - More data items in one set
    - Less “collisions” within set
    - Final comparison (matching tags?) gate delay?
- Maximum (\( v \) of cache lines)
  - \( \Rightarrow \) fully associative mapping
    - Whole cache is one set!
- Minimum (1)
  - \( \Rightarrow \) direct mapping
    - Each cache line is a set!
Cache Replacement Algorithm

- Which cache block to replace to make room for new block from memory?
- Direct mapping: trivial
- First-In-First-Out (FIFO)?
- Least-Frequently-Used (LFU)?
- Random?
- Which one is best / possible?
  - Chip area?
  - Fast? Easy to implement?

Cache Write Policy – memory writes?

- Write through (läpikirjoittava)
  - Each write goes always to cache and memory
  - Each write is a cache miss!
  - Write back (lopukäs/takaisin kirjoittava)
  - Each write goes only to cache
  - Write cache block back to memory
  - only when it is replaced in cache
  - Memory may have stale (old) data
  - cache coherence problem (eheyys, yhdenmukaisuus)
  - Write once ("vain kerran kirjoittava?")
  - Write-invalidate Snoopy-cache coherence protocol for multiprocessors
  - Write invalidates data in other caches
  - Write to memory at replacement time, or when some other cache needs it (has read/write miss)

Cache Line Size

- How big cache line?
- Optimise for temporal or spatial locality?
  - bigger cache line → better for spatial locality
  - more cache lines → better for temporal locality
- Best size varies with program or program phase?
- Best size different with code and data?
- 2-8 words?
  - word = 1 float??

Types and Number of Caches

- Same cache for data and code, or not?
  - Data references and code references behave differently
  - Unified vs. split cache (yhdistetty/erilliset)
  - Split cache: can optimise structure separately for data and code
- One cache too large for best results?
  - “Smaller is faster”
- Multiple levels of caches
  - L1 on same chip as CPU
  - L2 on same package or chip as CPU
    - older systems: same board
  - L3 on same board as CPU

Example: Pentium 4 Block Diagram

- L1: split, 4-way set-associative, line size 64 B
- L2, L3: unified, 8-way set-associative, line size 128 B

Main Memory
**Main Memory Types**

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Category</th>
<th>Error</th>
<th>Write Mechanism</th>
<th>Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random access semiconductor</td>
<td>ROM</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
</tr>
<tr>
<td>RAM</td>
<td>DRAM</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
</tr>
<tr>
<td>DRAM</td>
<td>SRAM</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
</tr>
<tr>
<td>SRAM</td>
<td>SDRAM</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
</tr>
<tr>
<td>SDRAM</td>
<td>RDRAM</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
<td>Non-volatile</td>
</tr>
</tbody>
</table>

**Random access semiconductor memory**

- Direct access to each memory cell
- Access time same for all cells

**RAM**

- Dynamic RAM, DRAM
  - Periodic refreshing required
  - Refresh required after read
  - Simpler, slower, denser, bigger (bytes per chip)
  - Access time ~ 60 ns (?)
  - Main memory? (early systems)

- Static RAM, SRAM
  - No periodic refreshing needed
  - Data remains until power is lost
  - More complex (more chip area/byte), faster, smaller
  - Access time ~ 25 ns (?)
  - Cache?

**SDRAM (Synchronous DRAM)**

- CPU clock synchronizes also the bus
- Runs on higher clock speeds than ordinary DRAM
- CPU knows how long it takes to make a reference,
  - can do other work while waiting
- 16 bits in parallel
  - Access 4 DRAMs (4 bits each) in parallel
  - Access time ~ 16 ns, transfer rate ~ 1.3 GB/s

- DDR SDRAM, double data rate
  - Current main memory technology
  - Supports transfers both on rising and falling edge of the clock cycle
  - Consumes less power
  - Access time ~ 12 ns, transfer rate ~ 3.2 GB/s

**Rambus DRAM (RDRAM)**

- Works with fast Rambus memory bus (800Mbps)
  - Controller + RDRAM modules
  - Access time ~ 12 ns, transfer rate ~ 4.8 GB/s
  - Speed slows down with many memory modules
  - Serially connected on Rambus channel

---

Comp. Org II, Autumn 2010

5.11.2010
MRAM

- Magnetoresistive Random Access Memory (MRAM)
  - Data stored with magnetic fields on two plates
  - Magnetic field directions determine bit value
- Non-volatile, data remains with power off
  - Fast to read/write
  - No upper limit for write counts (Flash has upper limit)
  - Access time comparable to DRAM, almost as fast as SRAM
- Future open
  - Small market share now
  - Still under development
  - May replace flash "in a few years" - HAS not by 2010 ...

Summary

- Memory hierarchy
- Cache
  - Size, Line size, Mapping, nr of levels, unified or split, write policy, replacement policy
- Memory
  - DRAM, SRAM
  - Overall features, no details required
  - Current technologies
    - Synchronous DRAM (SDRAM)
    - Double Data Rate DRAM (DDR)
    - Rambus DRAM

Review questions

- Memory hierarchy and principle of locality?
- Different ways to use locality in cache solutions?
- Differences of associative and set associative mappings?
- Why to have separate caches for instructions and data?