Internal Memory

Cache

Ch 4, Ch 5 [Sta10]

Key Characteristics
Locality
Cache
Main Memory
## Key Characteristics of Memories / Storage

<table>
<thead>
<tr>
<th>Location</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</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>

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Physical Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word size</td>
<td>Semiconductor</td>
</tr>
<tr>
<td>Number of words</td>
<td>Magnetic</td>
</tr>
<tr>
<td></td>
<td>Optical</td>
</tr>
<tr>
<td></td>
<td>Magneto-Optical</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unit of Transfer</th>
<th>Physical Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word</td>
<td>Volatile/nonvolatile</td>
</tr>
<tr>
<td>Block</td>
<td>Erasable/nonerasable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Method</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td></td>
</tr>
<tr>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td></td>
</tr>
<tr>
<td>Associative</td>
<td></td>
</tr>
</tbody>
</table>
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)

up: smaller, faster, more expensive, more frequent access
down: bigger, slower, less expensive, less frequent access

(Sta10 Fig 4.1)
Principle of locality (*paikallisuus*)

- In any given time period, memory references occur only to a small subset of the whole address space.

  = The reason why memory hierarchies work.

\[
\text{Prob (small data set)} = 95\%
\]
\[
\text{Prob (the rest)} = 5\%
\]
\[
\text{“Cost” (small data set)} = 0.01 \, \mu s
\]
\[
\text{“Cost” (the rest)} = 0.1 \, \mu s
\]

\[
\text{Aver cost} = 95\% \times 0.01 \, \mu s + 5\% \times 0.1 \, \mu s = 0.015 \, \mu s
\]

- Average cost is close to the cost of small data set.
- How to determine data for that small set?
- How to keep track of it?
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 items close to the one referenced a short time ago will be referenced soon

MEM: 345 23 71 8 305 63 91 2
Cache
Teemu’s Cheesecake

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

<table>
<thead>
<tr>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>1 sec</td>
</tr>
<tr>
<td>Hand</td>
<td>0.5 sec</td>
</tr>
<tr>
<td>Refrigerator</td>
<td>10 sec</td>
</tr>
<tr>
<td>Moon (Jupiter)</td>
<td>12 days</td>
</tr>
<tr>
<td>Europa (Jupiter)</td>
<td>4 years</td>
</tr>
</tbody>
</table>

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

Word Transfer

Block Transfer

(lohko, rivi)

Cache

Memory

Sta10 Fig 4.3, 4.4
Cache Read

(RA = Real Address, Main memory address)

START

Receive address RA from CPU

Is block containing RA in cache?

No

“Miss”

Access main memory for block containing RA

Write "dirty" cache line back to memory?

Yes

“Hit”

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

(Sta10 Fig 4.5)
Cache Organization
## Cache Design

**Cache Size & Line Size**

- Many blocks help for temporal locality
- Large blocks help for spatial locality
- Larger cache is slower
- Multi-level cache

### Cache Size & Write Policy

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Mapping Function</th>
<th>Write Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Direct</td>
<td>Write through</td>
</tr>
<tr>
<td></td>
<td>Associative</td>
<td>Write back</td>
</tr>
<tr>
<td></td>
<td>Set Associative</td>
<td>Write once</td>
</tr>
</tbody>
</table>

### Replacement Algorithm & Line Size

<table>
<thead>
<tr>
<th>Replacement Algorithm</th>
<th>Number of caches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Least recently used (LRU)</td>
<td>Single or two level</td>
</tr>
<tr>
<td>First in first out (FIFO)</td>
<td>Unified or split</td>
</tr>
<tr>
<td>Least frequently used (LFU)</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td></td>
</tr>
</tbody>
</table>

### Typical sizes:

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

*(Sta10 Table 4.1)*
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 (*täysin assosiaatiivinen*)
  - Any possible location
- Set associative mapping (*joukkoassosiaatiivinen*)
  - Some possible locations

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

34 bit address (byte address)

<table>
<thead>
<tr>
<th>tag</th>
<th>index</th>
<th>byte offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

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

Fixed location in cache → fixed cache size

Block number (in memory)

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

Fixed location in cache

= 28 = 256 blocks = 8 KB

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

Cache line size =

0x2480
0x6480
0xA480

Fixed location in cache

= 28 = 256 blocks = 8 KB
Direct Mapping Example

- Word = 4B (here)
- Block size = $2^3 = 8$ bytes = 64 bits
- Cache line size (for dir.mapping)

**ReadW I2, 0xA4**

- 8 bit address (byte address)
- Tag index: 2, 3
- Offset: 3

```
01 54 A7 00 91 23 66 32 11
11 77 55 55 66 66 22 44 22
01 65 43 21 98 76 65 43 32
```

- No match
- Compare

Read new memory block from memory address 0xA0=1010 0000 to cache location 100, update tag, and then continue with data access.
Direct Mapping Example 2

ReadW 12, 0xB4

tag 2
index 3
offset 3

Cache

tag 2
block 64

000:
001:
010:
011: 01 54 A7 00 91 23 66 32 11
100: 11 77 55 55 66 66 22 44 22
101: 01 65 43 21 98 76 65 43 32
110: 10 00 11 22 33 44 55 66 77
111:
Fully Associative Mapping

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

34 bit address (byte address)

Offset from the beginning of the block (in bytes)
Block size $= 2^5 = 32$ B

Block number (in memory)

Unique bits that are different for each block

Each block can be anywhere

Cache size can be any number of blocks

Alpha AXP uses 34 bit memory addresses

Computer Organization II, Spring 2012, Tiina Niklander
Fully Associative Example

ReadW I2, 0xB4

Cache

<table>
<thead>
<tr>
<th>tag (5)</th>
<th>block (64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:</td>
<td>12 34 56 78 9A 01 23 45</td>
</tr>
<tr>
<td>001:</td>
<td>87 00 32 89 65 A1 B2 00</td>
</tr>
<tr>
<td>010:</td>
<td>87 54 00 89 65 A1 B2 00</td>
</tr>
<tr>
<td>011:</td>
<td>54 A7 00 91 23 66 32 11</td>
</tr>
<tr>
<td>100:</td>
<td>77 55 55 66 66 22 44 22</td>
</tr>
<tr>
<td>101:</td>
<td>65 43 21 98 76 65 43 32</td>
</tr>
<tr>
<td>110:</td>
<td>00 11 22 33 44 55 66 77</td>
</tr>
<tr>
<td>111:</td>
<td>87 54 32 89 65 A1 B2 00</td>
</tr>
</tbody>
</table>

Match

Parallel!
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?
    - \( \log_2(64) = 6 \) 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 (set) contains 2 blocks
  - Use set (set index) field to find the cache entry
  - Use tag to determine if the block is in the set and where in it
  - Use offset to find the proper byte in the block

- 34 bit address (byte address)
  - Tag: 22
  - Set: 7
  - Offset: 5

- Block size $= 2^5 = 32$ B

- Unique bits that are different for each block, stored with block

- Nr of sets $= v = 2^7 = 128$ blocks $= 4$ KB

- Total cache size $= k*v = 2*4$ KB $= 8$ KB (without tag bits!)
2-way Set Associative Cache

- $k=2 \rightarrow$ two blocks in each set (\(= \) in one cache entry)
- 2 words in a block = 8 bytes \(\rightarrow\) 3 bits for byte offset
- Cache size 16 words = 4 sets \(\rightarrow\) 2 bits for set index
- Remaining 3 bits for tag

<table>
<thead>
<tr>
<th>set</th>
<th>tag</th>
<th>block</th>
<th>tag</th>
<th>block</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>110</td>
<td>12 34 56 78 9A</td>
<td>011</td>
<td>77 55 55 66 66</td>
</tr>
<tr>
<td>01</td>
<td>110</td>
<td>87 00 32 89 65</td>
<td>101</td>
<td>65 43 21 98 76</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>87 54 00 89 65</td>
<td>101</td>
<td>00 11 22 33 44</td>
</tr>
<tr>
<td>11</td>
<td>101</td>
<td>54 A7 00 91 23</td>
<td>111</td>
<td>00 11 22 33 44</td>
</tr>
</tbody>
</table>

8 bit address
(by byte address)

Discussion?
2-way Set Assoc. Cache Example

ReadW I2, 0xB4

Parallel! Match
Set Associative Mapping

- Set associative cache with set size \( k=2 \)
  \[ = 2\text{-way cache (common)} \]

- Degree of associativity = nr of blocks in a set = \( v \)
  - Large degree of associativity?
    - More data items in one set
    - Less “collisions” within set
    - Final comparison (matching tags?) gate delay?

- Maximum (nr of cache lines)
  \[ \Rightarrow \text{fully associative mapping} \]
  \[ \text{Whole cache is one set!} \]

- Minimum (1)
  \[ \Rightarrow \text{direct mapping} \]
  \[ \text{Each cache line is a set!} \]
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** (läpi kirjoittava)
  - Each write goes always to cache and memory
  - Each write is a cache miss!

- **Write back** (lopuksi/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 (eheys, yhdenmukaisuus)

Coherence problems:
- More users of the same data: memory valid? cache valid?
- Multiple processors with own caches

- **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)

Discussion?
Cache Line Size

- How big cache line?
- Optimise for temporal or spatial locality?
  - bigger cache line \( \Rightarrow \) better for spatial locality
  - more cache lines \( \Rightarrow \) 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 chip as CPU
    - Older systems: same board
  - L3 on same board as CPU
    - Newer systems: same chip
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

(Sta10 Fig 4.13)
Main Memory
### Main Memory Types

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Category</th>
<th>Erasure</th>
<th>Write Mechanism</th>
<th>Volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random-access memory (RAM)</td>
<td>Read-write memory</td>
<td>Electrically, byte-level</td>
<td>Electrically</td>
<td>Volatile</td>
</tr>
<tr>
<td>Read-only memory (ROM)</td>
<td>Read-only memory</td>
<td>Not possible</td>
<td>Masks</td>
<td></td>
</tr>
<tr>
<td>Programmable ROM (PROM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erasable PROM (EPROM)</td>
<td>Read-mostly memory</td>
<td>UV light, chip-level</td>
<td>Electrically</td>
<td>Nonvolatile</td>
</tr>
<tr>
<td>Electrically Erasable PROM (EEPROM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash memory</td>
<td></td>
<td></td>
<td></td>
<td></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?

Analog: Charge on capacitors

Digital: flip-flop gates
DRAM Access, 16 Mb DRAM (4M x 4)

22 bit address
- row access select (RAS)
- column access select (CAS)
- interleaved on 11 address pins

Discussion?

(Sta10 Fig 5.3)
256-KB DRAM Memory Organization

- Simultaneous access to 256K 8-bit word memory chip to access larger data items
- Access 64-bit words in parallel?
  - Need 8 chips.

(Sta10 Fig 5.5)
**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 ~ 18 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

(Sta10 Fig 5.14)
Example of alternative storage technology: 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, comparable to DRAM
  - No upper limit for write counts (Flash has upper limit)

- Future open
  - Small market share
  - Still under development

- Other new ideas:
  - FeRAM – ferromagnetic layer
  - PRAM or C-RAM – chalcogenide glass
  - NRAM – carbon nanotubes

Figure origin: http://www.research.ibm.com/journal/rd/501/maffitt.html (no longer available)
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?