

Cache

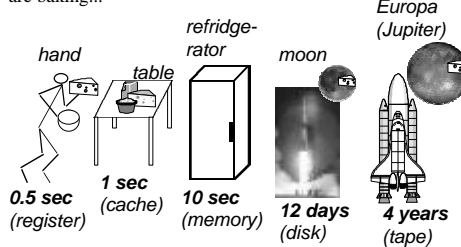
Ch 4.1-3

Memory Hierarchy
Main Memory
Cache
Implementation

10.11.1999 Copyright Teemu Kerola 1999 1

HW Speed Parallel

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



10.11.1999 Copyright Teemu Kerola 1999 2

Goal (4)

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

cache
HW solution
virtual memory
HW help for SW solution

10.11.1999 Copyright Teemu Kerola 1999

Memory Hierarchy (5)

- Most often needed data is kept close
- Access to small data sets can be made fast – simpler circuits
- Faster is more expensive
- Large can be bigger and cheaper

Memory Hierarchy

up: smaller, faster, more expensive, more frequent access

down: bigger, slower, less expensive, less frequent access

Fig. 4.1

10.11.1999 Copyright Teemu Kerola 1999 4

Locality (5)

(paikallisuus)

Temporal locality
data referenced again soon (ajallinen paikallisuus)

Spatial locality
nearby data referenced soon (alueellinen paikallisuus)

- The reason why memory hierarchies work

Prob (small data set) = 99%	Cost (small data set) = 2 μs
Prob (the rest) = 1%	Cost (the rest) = 20 μs

Aver cost $99\% * 2 \mu s + 1\% * 20 \mu s = 2.2 \mu s$

Close to cost of small data set

10.11.1999 Copyright Teemu Kerola 1999 5

Memory

- Random access semiconductor memory
 - give address & control, read/write data
- ROM, PROMS
 - system startup memory, BIOS (Basic Input/Output System)
 - load and execute OS at boot
 - also random access
- RAM
 - “normal” memory accessible by CPU

Table 4.2

10.11.1999 Copyright Teemu Kerola 1999 6

RAM

- **Dynamic RAM, DRAM** \$5 / MB? (1997)
 - simpler, slower, denser, bigger (?)
 - main memory? E.g., 60 ns access
 - periodic refreshing required
- **Static RAM, SRAM** \$100 / MB?
 - more complex, faster, smaller E.g., 5 ns access?
 - cache?
 - no periodic refreshing needed
 - data remains until power is lost

10.11.1999
Copyright Teemu Kerola 1999
7

DRAM Access

- **16 Mb DRAM**
 - 4 bit data items Fig. 4.4 Fig. 4.5 (b)
 - 4M data elements, 2K * 2K square
 - Address 22 bits
 - row access select (RAS)
 - column access select CAS
 - interleaved on 11 address pins
- **Simultaneous access to many 16Mb memory chips to access larger data items**
 - Access 32 bit words in parallel? Need 8 chips.

10.11.1999
Copyright Teemu Kerola 1999
8

10.11.1999
Copyright Teemu Kerola 1999
9

Cache Memory (välimuisti)

- **Problem: how can I make my (main) memory as fast as my registers?**
- **Answer: (processor) cache**
 - keep most probably referenced data in fast cache close to processor, and rest of it in memory
 - much smaller than main memory
 - (much) more expensive (per byte) than memory
 - most of data accesses to cache 90% 99%?

10.11.1999
Copyright Teemu Kerola 1999
10

Cache Operation ⁽⁵⁾

- **Data is in cache?** Hit Fig. 4.15
- **Data is only in memory?** Miss

Read it to cache
CPU waits until data available

Many blocks help for temporal locality
many different data items in cache Fig. 4.14

Large blocks help for spatial locality
lots of "nearby" data available

Fixed cache size?
Select "many" or "large"?

10.11.1999
Copyright Teemu Kerola 1999
11

Cache Features

- **Size**
- **Mapping function** (kuvausfunktio)
 - how to find data in cache?
- **Replacement algorithm** (poistoalgoritmi)
 - which block to remove to make room for a new block?
- **Write policy** (kirjoituspolitiikka)
 - how to handle writes?
- **Line size (block size)?** (rivin tai lohkon koko)
- **Number of caches?**

10.11.1999
Copyright Teemu Kerola 1999
12

Cache Size

- Bigger is better in general
- Bigger may be slower
 - lots of gates, cumulative gate delay?
- Too big might be too slow!
 - Help: 2- or 3-level caches

1KW (4 KB), 512KW (2 MB)?

10.11.1999 Copyright Teemu Kerola 1999 13

Mapping: Memory Address (3)

- Alpha AXP issues 34 bit memory addresses
 - At cache hit block offset is controlling a multiplexer to select right word

34 bit address (byte address)

block address

block offset

29 bits ← 5 bits ←

max physical address space = $2^{34} = 16GB$

Cache line size = block size = $2^5 = 32$ bytes = 4 words

Number of possible blocks in physical address space = $2^{29} = 500M$ blocks

10.11.1999 Copyright Teemu Kerola 1999 14

Direct Mapping (6) (suora kuvaus)

- Every block has only one possible location (cache line number) in cache
 - determined by index field bits

34 bit address (byte address)

Block address (in memory)

Cache line size = block size = $2^5 = 32$ bytes

tag index offset

21 bits 8 bits 5 bits

Unique bits that are different for each block, stored in each cache line

Addr in cache, cache size = $2^8 = 256$ blocks = 8 KB

Fig. 4.17 (s = 29, r = 8, w = 5)
Fig. 7.10 from [PaHe98]

10.11.1999 Copyright Teemu Kerola 1999 15

Direct Mapping Example (5)

Word = 4 bytes (here)

ReadW I2, 0xA4
0xA4 = 1010 0100

8 bit address (byte address)

tag index offset

Cache line size = block size = $2^3 = 8$ bytes = 64 bits

tag index offset

2 3 3

tag	index	offset	data
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:			

No match

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

10.11.1999 Copyright Teemu Kerola 1999 16

Direct Mapping Example 2 (5)

ReadW I2, 0xB4
0xB4 = 1011 0100

8 bit address (byte address)

tag index offset

Cache line size = block size = $2^5 = 32$ bytes

tag index offset

2 3 3

tag	index	offset	data
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:			

Match

10.11.1999 Copyright Teemu Kerola 1999 17

Fully Associative Mapping (5) (täysin assosiativinen kuvaus)

- Every block can be in any cache line
 - tag must be complete block address

34 bit address (byte address)

Block address (in memory)

Cache line size = block size = $2^5 = 32$ bytes

tag block offset

29 bits 5 bits

Unique bits that are different for each block

Cache line can be anywhere
Cache size can be any number of blocks

Fig. 4.19 (s = 29, w = 5)

10.11.1999 Copyright Teemu Kerola 1999 18

Fully Associative Mapping

- Lots of circuits
 - tag fields are long - wasted space!
 - each cache line tag must be compared simultaneously with the memory address tag
 - lots of wires
 - lots of comparison circuits
- Final comparison "or" has large gate delay
 - did any of these 64 comparisons match?
 - $2^{\log(64)} = 8$ levels of binary gates
 - how about 262144 comparisons? 18 levels?
- \Rightarrow Can use it only for small caches

Copyright Teemu Kerola 1999

Fully Associative Example (5)

ReadW I2, 0xB4
0xA4 = 1011 0100

tag	data
000: 11011	12 34 56 78 9A 01 23 45
001: 10111	87 00 32 89 65 A1 B2 00
010: 00011	87 54 00 89 65 A1 B2 00
011: 10100	54 A7 00 91 23 66 32 11
100: 00111	77 55 55 66 66 22 44 22
101: 10100	65 43 21 98 76 65 43 32
110: 10110	00 11 22 33 44 55 66 77
111: 10011	87 54 32 89 65 A1 B2 00

Copyright Teemu Kerola 1999

Set Associative Mapping (6)

(joukkoassosiatiivinen kuvaus)

- With set size $k=2$, every block has 2 possible locations (cache line number) in cache
 - location in each set determined by set (index) field bits

Cache line size = block size = $2^5 = 32$ bytes

34 bit address (byte address): tag (3), set (2), offset (5)

Unique bits that are different for each block, stored in each cache line

$v = 2^7$ sets, 128 blocks = 4 KB

Total cache size = $2 * 4$ KB = 8 KB

Fig. 5.8 from [HePa96]
Fig. 7.19 from [PaHe98]

Copyright Teemu Kerola 1999

2-way Set Associative Cache

Set	tag	data	1 st lines in each set
Set 0	00:	110	12 34 56 78 9A 01 23 45
Set 1	01:	110	87 00 32 89 65 A1 B2 00
Set 2	10:	100	87 54 00 89 65 A1 B2 00
Set 3	11:	101	54 A7 00 91 23 66 32 11
	00:	011	77 55 55 66 66 22 44 22
	01:	101	65 43 21 98 76 65 43 32
	10:	101	00 11 22 33 44 55 66 77
	11:	111	87 54 32 89 65 A1 B2 00

- 3 bit tag
- set size 2 \Rightarrow 2 cache lines per set
- 4 sets \Rightarrow 2 bits for set index
- 8 byte cache lines \Rightarrow 3 bits for byte address in cache line

Copyright Teemu Kerola 1999

Set Associative Example (6)

ReadW I2, 0xB4
0xB4 = 1011 0100

tag	set	offset	data
00:	110	12 34 56 78 9A 01 23 45	
01:	110	87 00 32 89 65 A1 B2 00	
10:	100	87 54 00 89 65 A1 B2 00	
11:	101	54 A7 00 91 23 66 32 11	
00:	011	77 55 55 66 66 22 44 22	
01:	101	65 43 21 98 76 65 43 32	
10:	101	00 11 22 33 44 55 66 77	
11:	111	87 54 32 89 65 A1 B2 00	

Copyright Teemu Kerola 1999

Set Associative Mapping

- Set associative cache with 2 sets = 2-way cache
- Degree of associativity v ? Usually 2
 - v large? Fig. 7.16 from [PaHe98]
 - More data items (v) in one set
 - less "collisions"
 - final comparison (matching tags?) gate delay?
 - v maximum (nr of cache lines) \Rightarrow fully associative mapping
 - v minimum (1) \Rightarrow direct mapping

Copyright Teemu Kerola 1999

Replacement Algorithm

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

10.11.1999 Copyright Teemu Kerola 1999 25

Write Policy

- How to handle writes to memory?
- Write through (läpikirjoittava)
 - each write goes always to memory
 - each write is a cache miss!
- Write back (lopuksi kirjoittava takaisin kirjoittava?)
 - write cache block to memory only when it is replaced in cache
 - memory may have stale (old) data
 - cache coherence problem (välimuistin yhteneväisyysongelma)

10.11.1999 Copyright Teemu Kerola 1999 26

Line size

- How big cache line?
- Optimise for temporal or spatial locality?
- Data references and code references behave in a different way
- Best size varies with program or program phase
- 2-8 words?
 - word = 1 float??

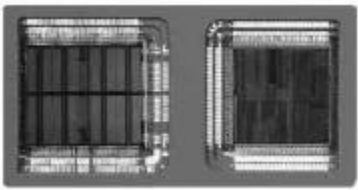
10.11.1999 Copyright Teemu Kerola 1999 27

Number of Caches ⁽³⁾

- One cache too large for best results
- Unified vs. split cache (yhdistetty, erilliset)
 - same cache for data and code, or not?
 - split cache: can optimise structure separately for data and code
- Multiple levels of caches
 - L1 - same chip as CPU
 - L2 - same package or chip as CPU
 - L3 - same board as CPU (Fig. 4.23)

10.11.1999 Copyright Teemu Kerola 1999 28

-- End of Ch. 4.3: Cache Memory --



<http://www.intel.com/procs/servers/feature/cache/unique.htm>

“The Pentium® Pro processor’s unique multi-cavity chip package brings L2 cache memory closer to the CPU, delivering higher performance for business-critical computing needs.”

10.11.1999 Copyright Teemu Kerola 1999 29