Lecture 6: EITF20 Computer Architecture

Anders Ardö

EIT – Electrical and Information Technology, Lund University

November 13, 2013

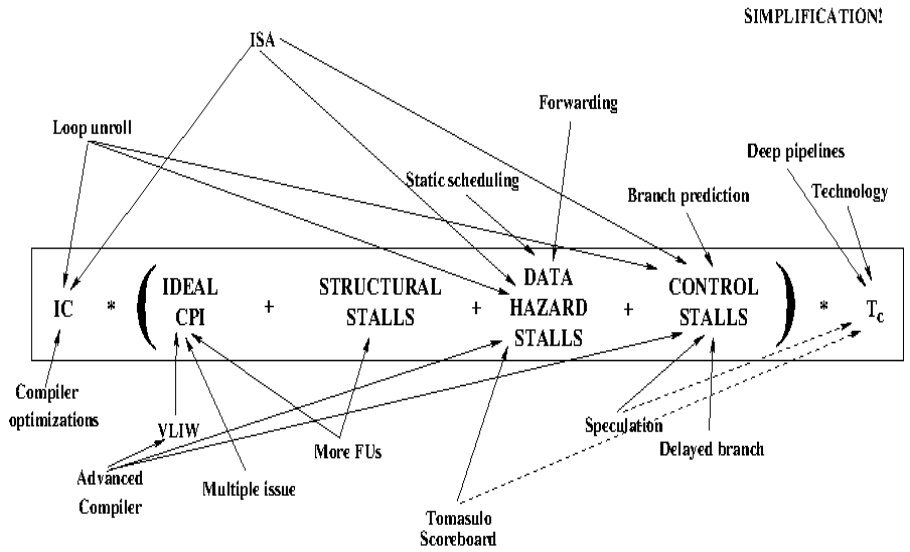
Outline

Dynamic scheduling, speculation - summary

- tolerates unpredictable delays
- compile for one pipeline - run effectively on another
- allows speculation
  - multiple branches
  - in-order commit
  - precise exceptions
  - time, energy; recovery
- significant increase in HW complexity
- out-of-order execution, completion
- register renaming

Tomasulo, CDB, ROB

CPU Performance Equation - Pipelining



Summary Pipelining - Implementation

Problem	Simple	Scoreboard	Tomasulo	Tomasulo + Speculation
	Static Sch	Dynamic Scheduling		
RAW	forwarding stall	wait (Read)	CDB stall	CDB stall
WAR	-	wait (Write)	Reg. rename	Reg. rename
WAW	-	wait (Issue)	Reg. rename	Reg. rename
Exceptions	precise	?	?	precise, ROB
Issue	in-order	in-order	in-order	in-order
Execution	in-order	out-of-order	out-of-order	out-of-order
Completion	in-order	out-of-order	out-of-order	out-of-order
Commit	in-order	out-of-order	out-of-order	in-order
Structural hazard	-	many FU stall	many FU, CDB, stall	many FU, CDB, stall
Control hazard	Delayed br., stall	Branch prediction	Branch prediction	Br. pred, speculation

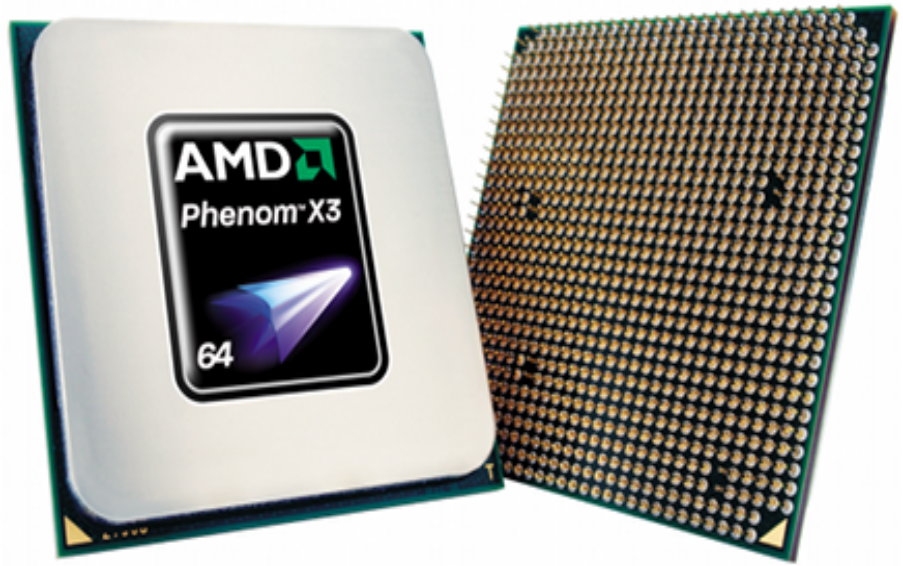
Getting CPI < 1!

- Issuing multiple instructions per clock cycle
  - ***Superscalar***: varying number of instructions/cycle (1-8) scheduled by compiler or HW
    - IBM Power5, Pentium 4, Core2, AMD, Sun SuperSparc, DEC Alpha
    - Simple hardware, complicated compiler or...
    - Complex hardware but simple for compiler
  - ***Very Long Instruction Word*** (VLIW): fixed number of instructions (3-5) scheduled by the compiler
    - HP/Intel IA-64, Itanium
    - Simple hardware, difficult for compiler
      high performance through extensive compiler optimization

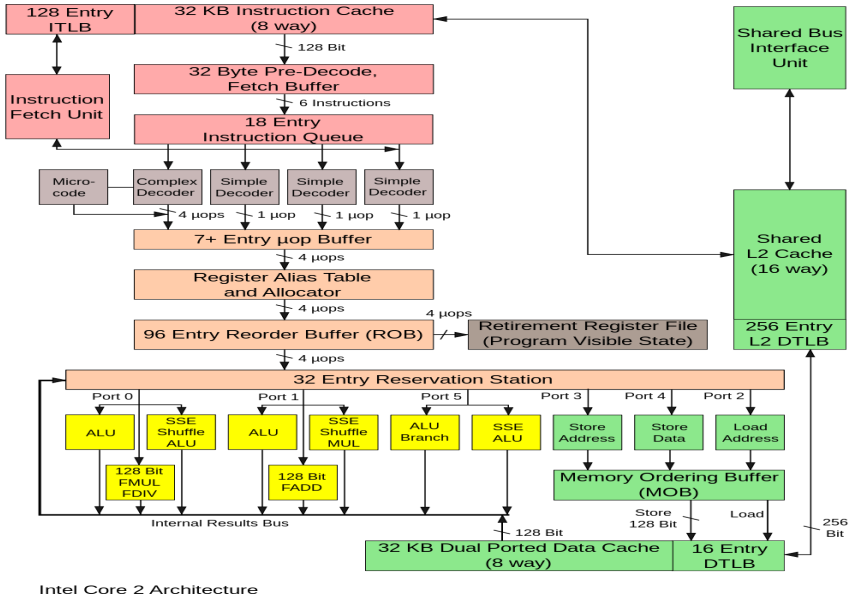
Approaches for multiple issue

	Issue	Hazard detection	Scheduling	Characteristics /examples
Superscalar	dynamic	HW	static	in-order execution ARM
Superscalar	dynamic	HW	dynamic	out-of-order execution
Superscalar	**dynamic**	**HW**	**dynamic**	**speculation** Pentium 4 IBM power5
VLIW	static	compiler	static	TI C6x
EPIC	static	compiler	mostly static	Itanium

AMD Phenom CPU

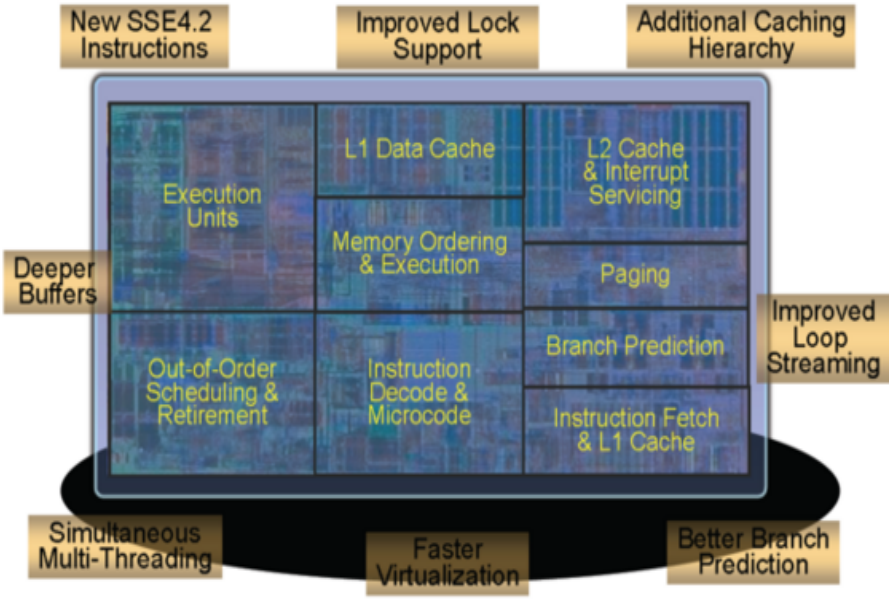


Intel Core2



Intel Core 2 Architecture

Intel Core2 chip (Nehalem)



Questions!

QUESTIONS?

COMMENTS?

Lecture 6 agenda

Chapters 5.1, 5.3 (and Appendix C.1-C.3) in "Computer Architecture"

1. Reiteration
2. Memory hierarchy
3. Cache memory
4. Cache performance
5. Main memory
6. Summary

Outline

Memory

You want a memory to be:

FAST
BIG
AND **CHEAP**

Contradiction!

How?

Memory - big, fast and cheap

Use two "magic" tricks

- Make a small memory seem bigger (Without making it much slower) — **virtual memory**
- Make a slow memory seem faster (Without making it smaller) — **cache memory**

Memory tricks (techniques)

Use a hierarchy

CPU	superfast	Registers	instructions	
Memory	FAST	Cache	cache memory	(HW)
	CHEAP	Main memory	virtual memory	(SW)
	BIG	Disk		

Levels of the Memory Hierarchy

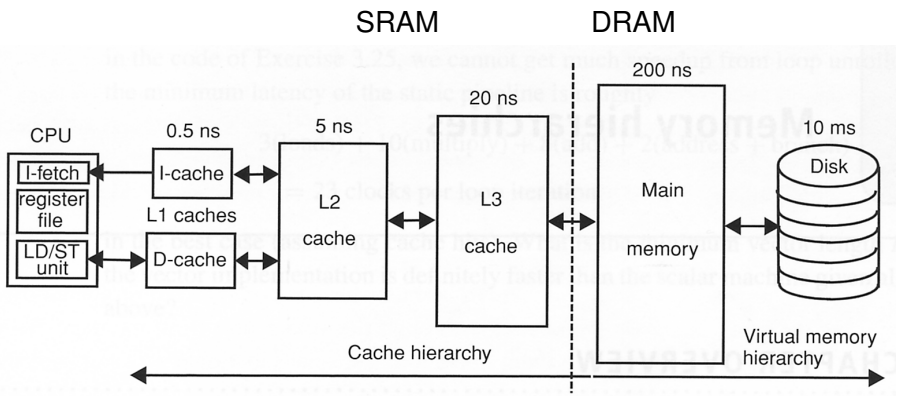
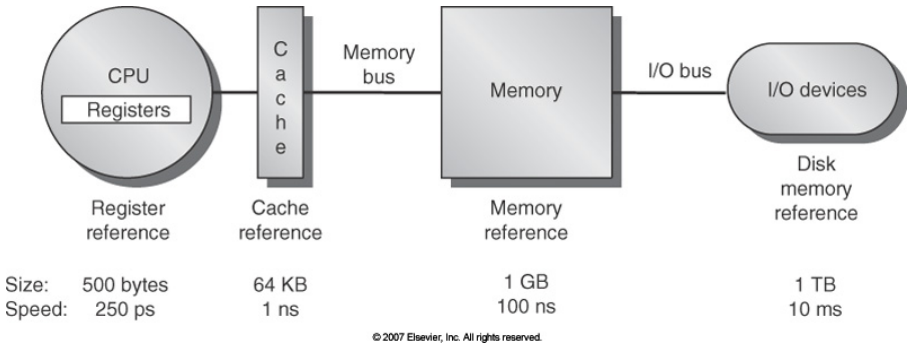


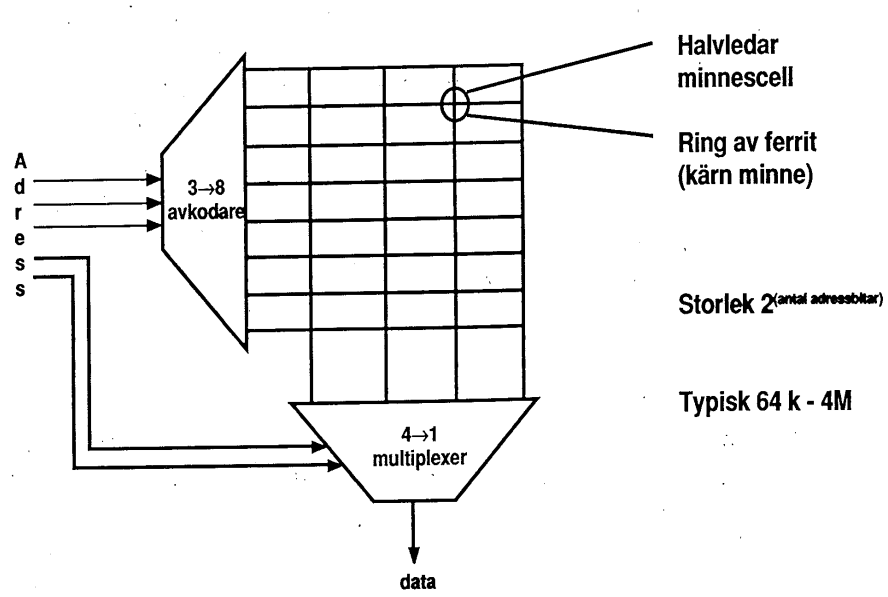
Figure 4.1. Typical memory hierarchy with three levels of caches.

(From Dubois, Annavaram, Stenström: "*Parallel Computer Organization and Design*", ISBN 978-0-521-88675-8)

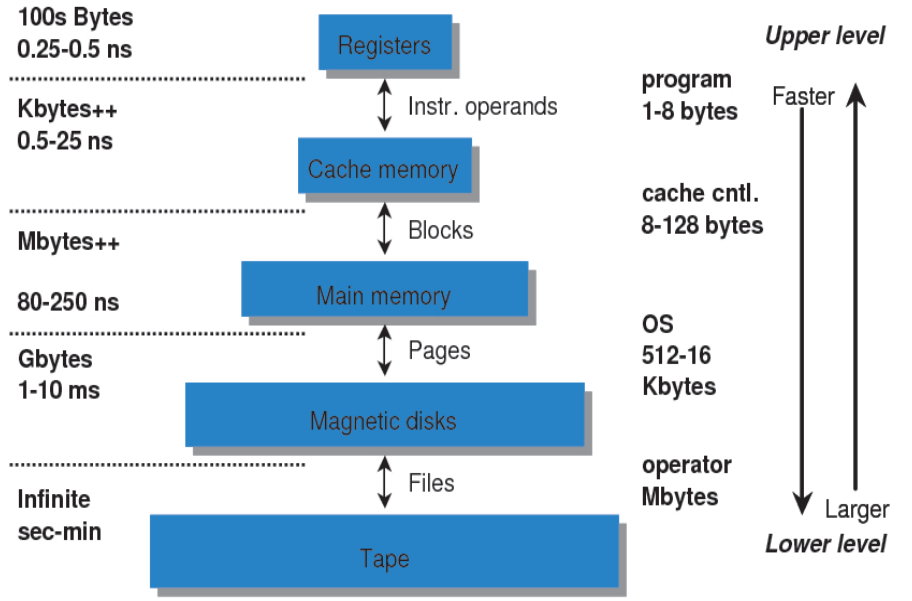
Initial model of Memory Hierarchy



RAM

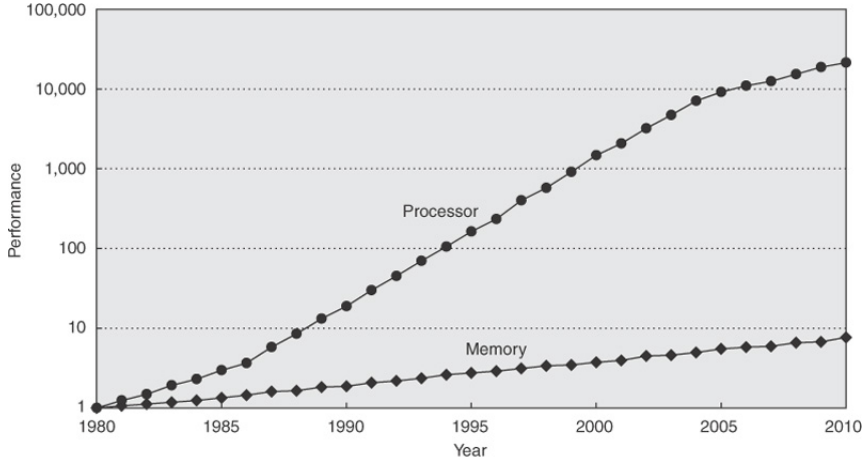


Levels of the Memory Hierarchy



Who cares?

- 1980: no cache in microprocessors
- 1995: 2-level caches in a processor package
- 2000: 2-level caches on a processor die
- 2003: 3-level caches on a processor die



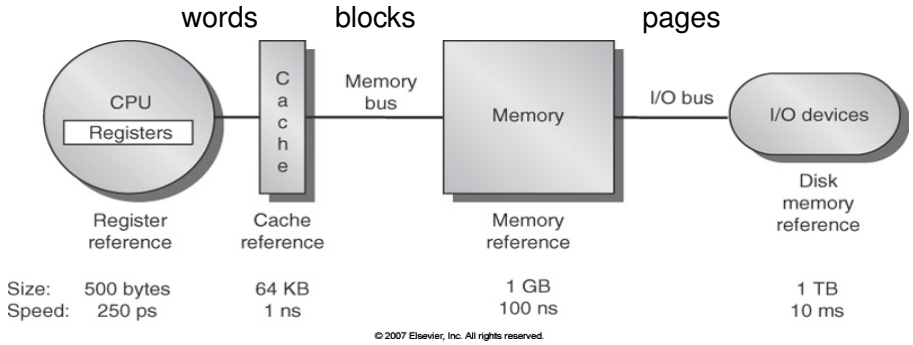
Example – bandwidth need

Assume an “ideal” CPU with no stalls, running at 3 GHz and capable of issuing 3 instructions (32 bit) per cycle.

Instruction fetch	4	bytes/instr
Load & store	1	byte/instr
Instruction frequency	$3 * 3 = 9$	GHz
Bandwidth	$9 * (4 + 1) = 45$	GB/sec

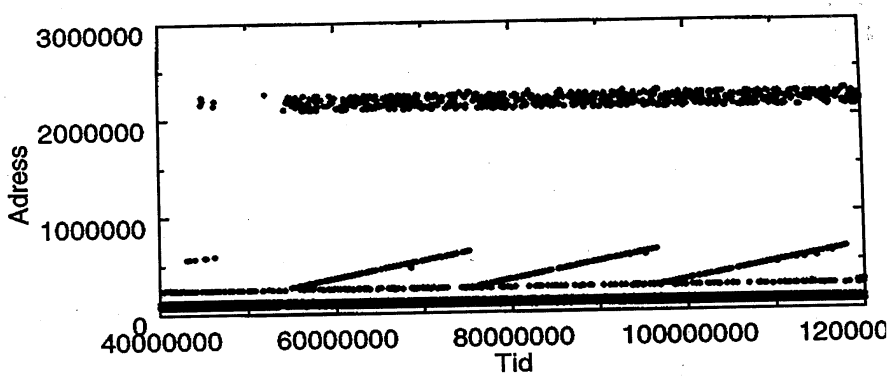
Memory Hierarchy Functionality

- CPU tries to access memory at address A. If A is in the cache, deliver it directly to the CPU
- If not – transfer a **block of memory words**, containing A, from the memory to the cache. Access A in the cache.
- If A not present in the memory – transfer a **page of memory blocks**, containing A, from disk to the memory, then transfer the block containing A from memory to cache. Access A in the cache.

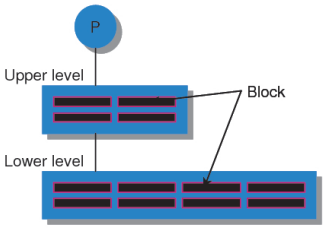


The Principle of Locality

- ***A program access a relatively small portion of the address space at any instant of time***
- Two different types of locality:
  - ***Temporal locality*** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - ***Spatial locality*** (Locality in space): If an item is referenced, items whose addresses are close, tend to be referenced soon.



Memory hierarchy – Terminology



- $\text{Size}_{\text{upper}} < \text{Size}_{\text{lower}}$
- $\text{Access time}_{\text{upper}} < \text{Access time}_{\text{lower}}$
- $\text{Cost}_{\text{upper}} > \text{Cost}_{\text{lower}}$

Upper level = cache; Lower level = Main memory

- ***Block***: The smallest amount of data moved between levels
- ***Hit***: A memory reference that is satisfied in the cache
- ***Miss***: A memory reference that *cannot* be satisfied in the cache

Outline

Cache measures

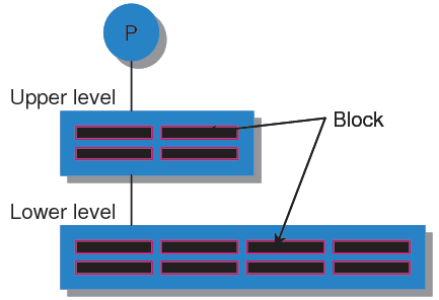
- hit rate = no of accesses that hit/no of accesses
  - close to 1, more convenient with
- miss rate = 1.0 − *hit rate*
- hit time: cache access time plus time to determine hit/miss
- miss penalty: time to replace a block
  - measured in ns or number of clock cycles

  depends on:
  - **latency**: time to get first word
  - **bandwidth**: time to transfer block

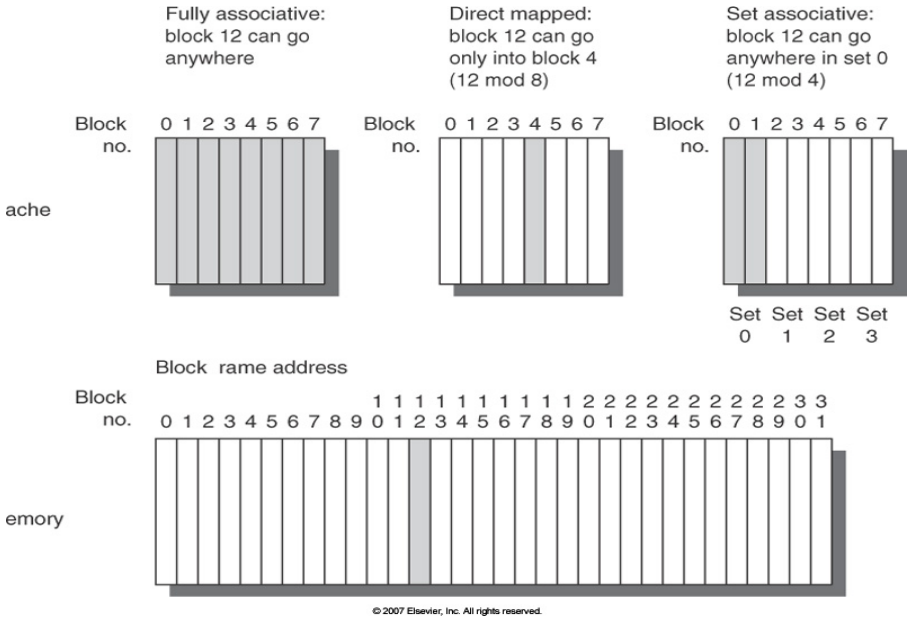
  out-of-order execution can hide some of the miss penalty
- Average memory access time
  $= \textit{hit time} + \textit{miss rate} * \textit{miss penalty}$

4 questions for the Memory Hierarchy

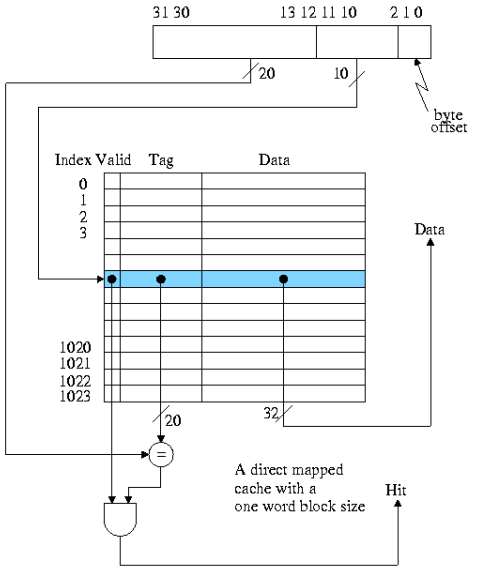
- Q1: Where can a block be placed in the upper level?
  (Block placement)
- Q2: How is a block found if it is in the upper level?
  (Block identification)
- Q3: Which block should be replaced on a miss?
  (Block replacement)
- Q4: What happens on a write?
  (Write strategy)



Block placement



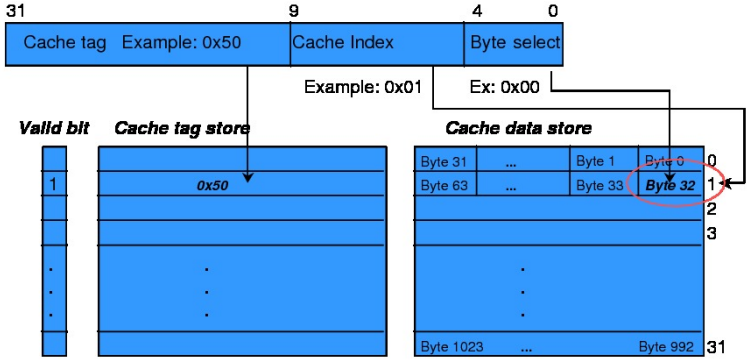
Block identification



Block identification

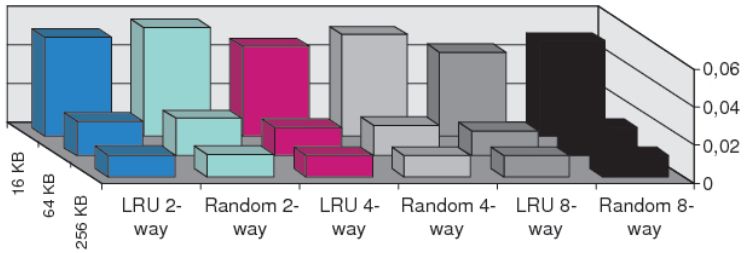
- Store a part of the address: ***Tag***

Example: a 1 KB, ***Direct Mapped Cache***, 32 byte blocks



Block replacement

- Direct mapped caches don't need a block replacement policy (why?)
- Primary strategies:
  - Random (easiest to implement)
  - LRU – Least Recently Used (best)
  - FIFO – Oldest



Cache read

- CPU tries to read memory address A
- Search the cache to see if A is there

hit	miss
⇓	Copy the block that contains A from lower-level memory to the cache. (Possibly replacing an existing block.)

- Send data to the CPU

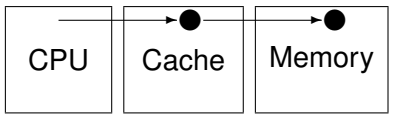
Write strategy

- ***Write through***: The information is written to both the block in the cache and to the block in the lower-level memory
  - Is always combined with write buffers so that the CPU doesn't have to wait for the lower level memory
- ***Write back***: The information is written only to the block in the cache.
  - Copy a modified cache block to main memory only when replaced
  - Is the block clean or modified? (**dirty bit**)
- Do we allocate a cache block on a write miss?
  - Write allocate (allocate a block in the cache)
  - No-write allocate (only write to memory)

Cache write

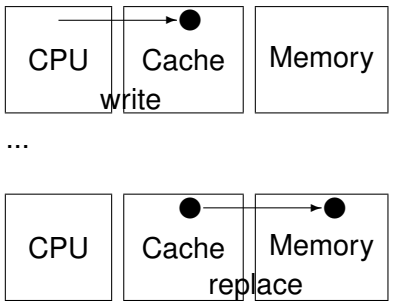
Write through

- Always write in both cache and lower-level memory

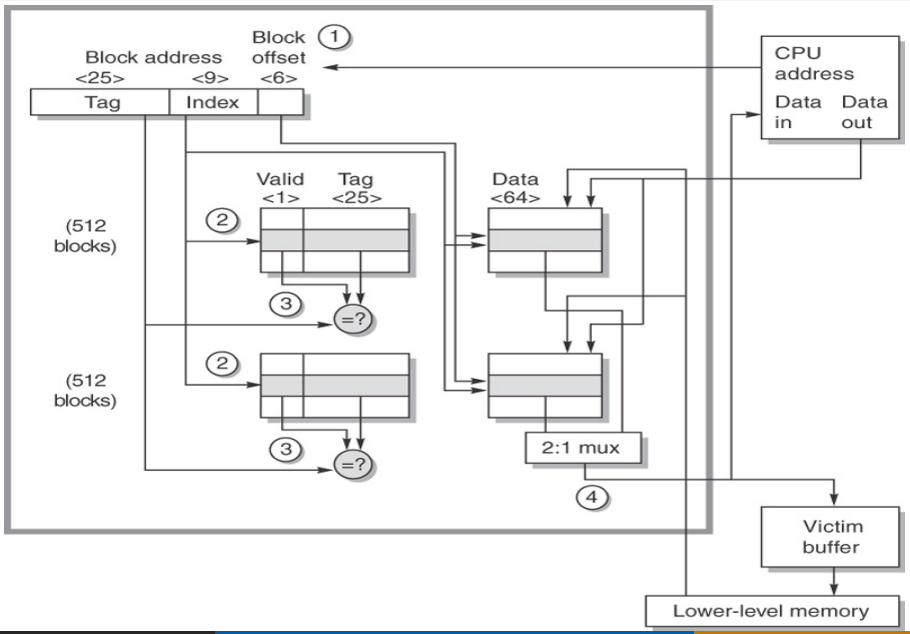


Write back

- Only write in cache
- Update lower-level memory when a block is replaced



Cache micro-ops sequencing



Cache micro-ops sequencing – read

- **1**: Address division
- **2**: Set/block selection
- **2a**: Tag read
- **3**: Tag/Valid bit checking
- **4**: Hit: Data out
  Miss: Signal cache miss; initiate replacement

Outline

Cache Performance

$$memory\ stall\ cycles_{cache}$$

$$= no\ of\ misses * miss\ penalty$$

$$= IC * \frac{misses}{instruction} * miss\ penalty$$

$$= IC * \frac{mem\ accesses}{instruction} * miss\ rate * miss\ penalty$$

CPU execution Time

$$= IC * CPI_{execution} * T_C + memory\ stall\ cycles_{cache} * T_C$$

$$= IC * (CPI_{execution} + \frac{mem\ accesses}{instruction} * miss\ rate * miss\ penalty) * T_C$$

Interlude - CPU Performance Equation

CPU execution Time =

$$IC * (CPI_{execution} + CPI_{cache}) * T_C =$$

$$IC * (CPI_{ideal} + CPI_{pipeline} + CPI_{cache}) * T_C =$$

$$IC * (CPI_{ideal} +$$

Structural Stalls + Data Hazard Stalls + Controll Stalls +

$$\frac{mem\ accesses}{instruction} * miss\ rate * miss\ penalty$$

$$) * T_C$$

Cache Performance

$$\text{CPU execution Time} = IC * (CPI_{execution} + \frac{\text{mem accesses}}{\text{instruction}} * \text{miss rate} * \text{miss penalty}) * T_C$$

Three ways to increase performance:

- Reduce miss rate
- Reduce miss penalty
- Reduce hit time (improves T_C)

Average memory access time =

$$\text{hit time} + \text{miss rate} * \text{miss penalty}$$

Performance aspects

$$\text{CPU execution Time} = IC * (CPI_{execution} + \frac{\text{mem accesses}}{\text{instruction}} * \text{miss rate} * \text{miss penalty}) * T_C$$

Example:

miss rate (%)	1
miss penalty (cycles)	50
$\frac{\text{mem accesses}}{\text{instruction}}$	k
CPI increase	$k * 0.01 * 50$

1 instruction fetch plus data accesses from ca 35 % of the instructions
$\Longrightarrow k \approx 1.35$;
assume $CPI_{execution} = 2.0$

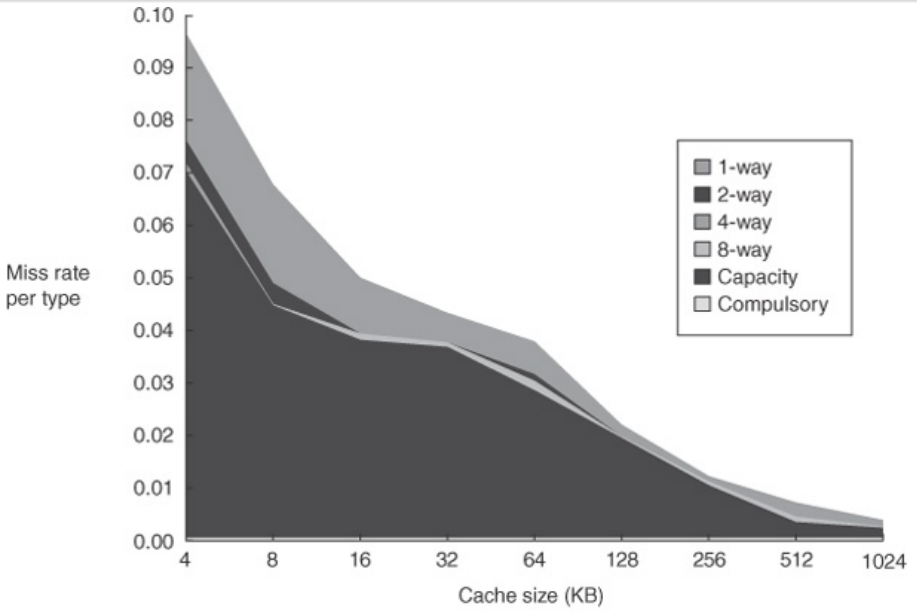
$$\frac{T_{exe\_ideal\_cache}}{T_{exe\_example\_cache}} = \frac{IC*2.0*T_C}{IC*(2.0+1.35*0.01*50)*T_C} \approx 0.75$$

Why does a memory reference miss?

- A cache miss can be classified as a:
  - ***Compulsory miss***: The first reference is always a miss
  - ***Capacity miss***: If the cache memory is to small it will fill up and subsequent references will miss
  - ***Conflict miss***: Two memory blocks may be mapped to the same cache block with a direct or set-associative address mapping

3 C's

Miss rate components – 3 C's

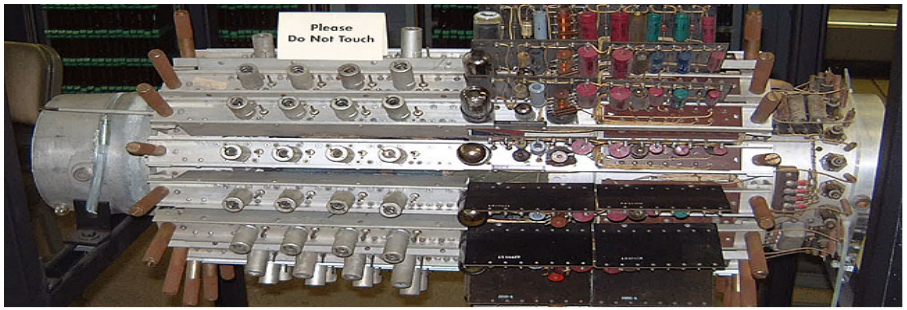


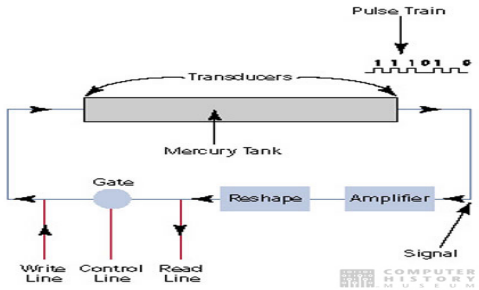
Miss rate components – 3 C's

- Small percentage of **compulsory** misses
- **Capacity** misses are reduced by larger caches
- full associativity avoids all **conflict** misses
- **Conflict** misses are relatively more important for small set-associative caches

Outline

Serial memory





Main memory: Background

- Performance of main memory:
  - ***Latency affects***: Cache miss penalty
    - *Access time*: time between request and word arrives
    - *Cycle time*: time between requests
  - ***Bandwidth affects***: I/O, multiprocessors (& cache miss penalty)
- Main memory is ***DRAM***: Dynamic RAM
  - Dynamic - memory cells need to be refreshed
  - 1 transistor and a capacitor per bit
- Cache memory is ***SRAM***: Static RAM
  - No refresh
  - 6 transistors per bit

SRAM technology

- Address typically not multiplexed
- Each cell consists of 6 transistors and a capacitor
- Fast access time
- Optimizes speed and capacity
- Expensive

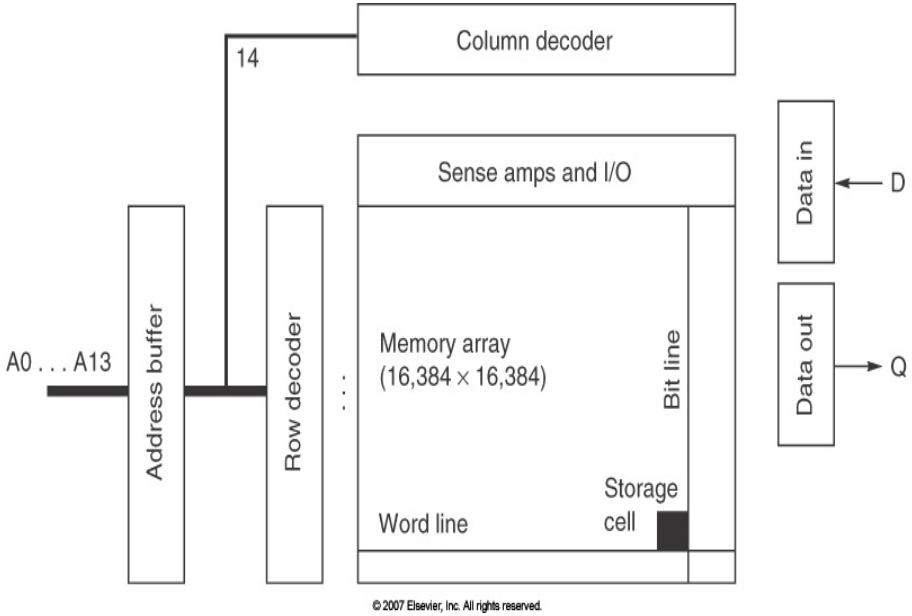
Used in caches

DRAM technology

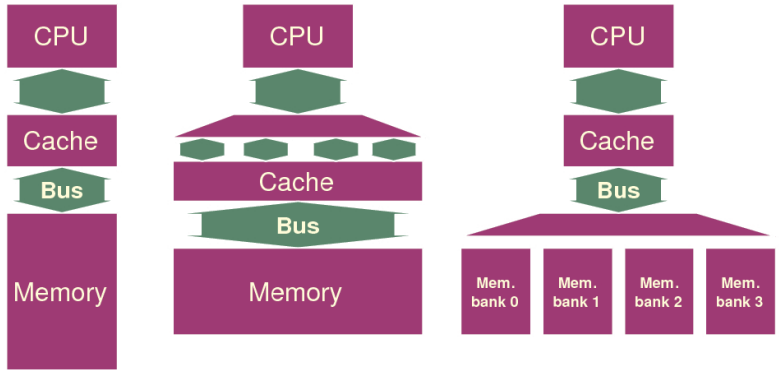
- The address is multiplexed in order to reduce the number of pins (increases latency)
- Refresh of memory cells decreases bandwidth
- The need of sense amplifiers increases latency
- Each cell consists of 1 transistor and 1 capacitor
- Optimizes cost/bit and capacity
- Internal optimizations: SDRAM, DDR, DDR2, DDR3

Used in main memory

DRAM organization



Improving main memory performance

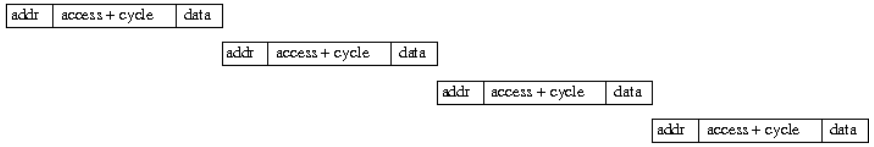


normal memory | wide memory | interleaving

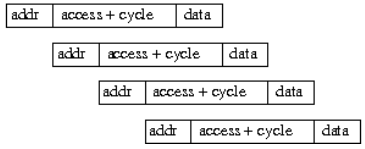
Improves bandwidth.

Memory interleave timing

Normal



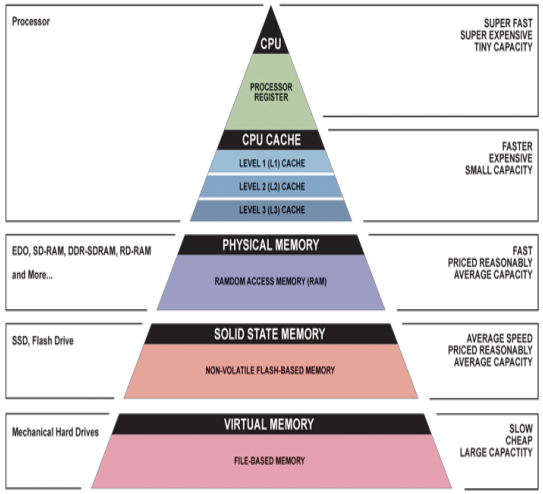
Interleaving 4–way



Outline

Summary

- The performance gap between CPU and memory is widening
- Memory Hierarchy
  - Cache level 1
  - Cache level ...
  - Main memory
  - Virtual memory
- Four design issues:
  - Block placement
  - Block identification
  - Block replacement
  - Write strategy



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Summary – cont

- Cache misses increases the CPI for instructions that access memory
- Three types of misses:
  - Compulsory
  - Capacity
  - Conflict

  3 C's
- Main memory performance:
  - Latency
  - Bandwidth (interleaving)