

# 8.8GHz Intel Core i9-13900K, Breaks World Record



Hardcore overclocker Elmor officially broke the CPU frequency world record with Intel's brand-new Core i9-13900K 24-core processor. The best CPU hit a staggering 8.812GHz using liquid nitrogen cooling, dethroning the 8-year reigning champion, the FX-8370, by 90MHz.

That's right; it took eight years for a new CPU architecture to dethrone AMD's FX series processors. Those chips are infamous for their mediocre CPU performance at launch; however, these chips scaled incredibly well under liquid nitrogen overclocking.

[www.tomshardware.com/news/i9-13900k-world-record-overclock](http://www.tomshardware.com/news/i9-13900k-world-record-overclock)



UC Berkeley  
Teaching Professor  
Dan Garcia

# CS61C

## Great Ideas in Computer Architecture (a.k.a. Machine Structures)



UC Berkeley  
Teaching Professor  
Lisa Yan

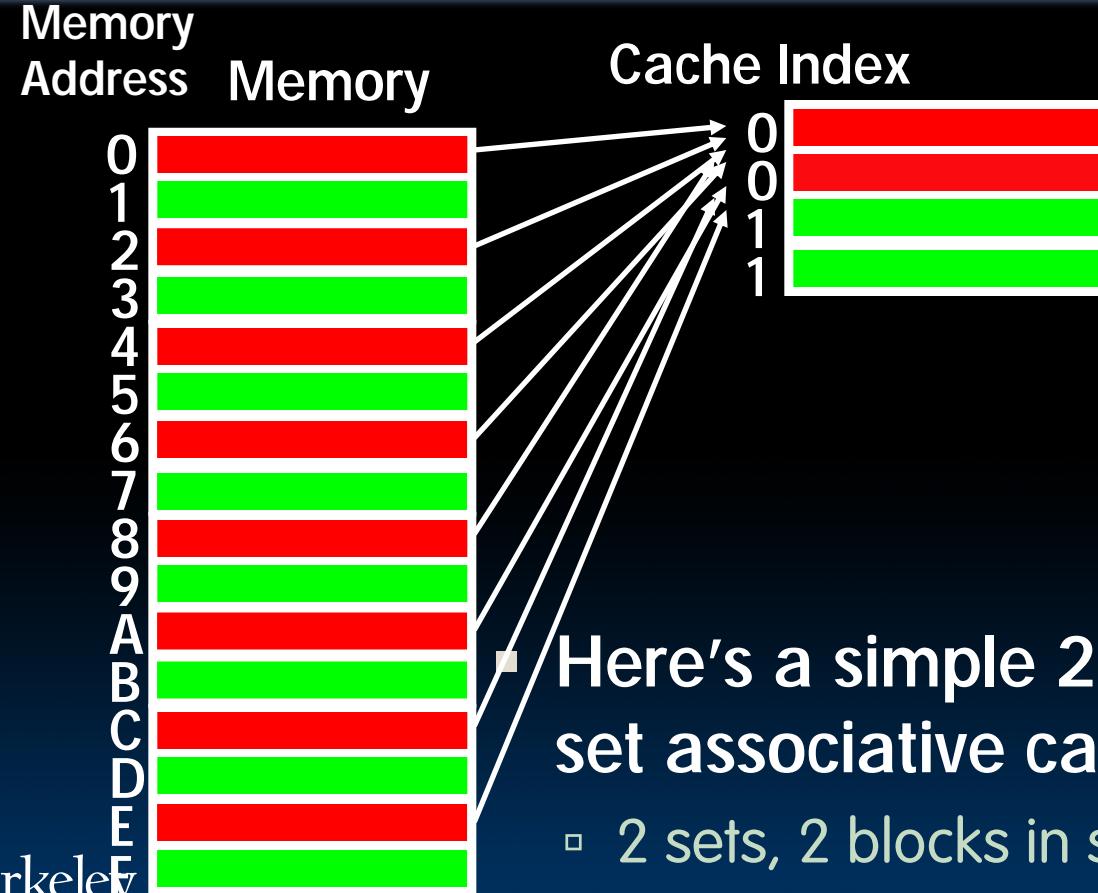
### Caches IV

# Set-Associative Caches

# N-Way Set Associative Cache (1/3)

- Memory address fields:
  - Tag: same as before
  - Offset: same as before
  - Index: points us to the correct “row” (called a set in this case)
- So what’s the difference?
  - each set contains multiple blocks
  - once we’ve found correct set, must compare with all tags in that set to find our data
  - Size of \$ is # sets  $\times$  N blocks/set  $\times$  block size

# Associative Cache Example



- Here's a simple 2-way set associative cache.

- 2 sets, 2 blocks in set

# N-Way Set Associative Cache (2/3)

- Basic Idea

- cache is direct-mapped w/respect to sets
  - each set is fully associative with N blocks in it

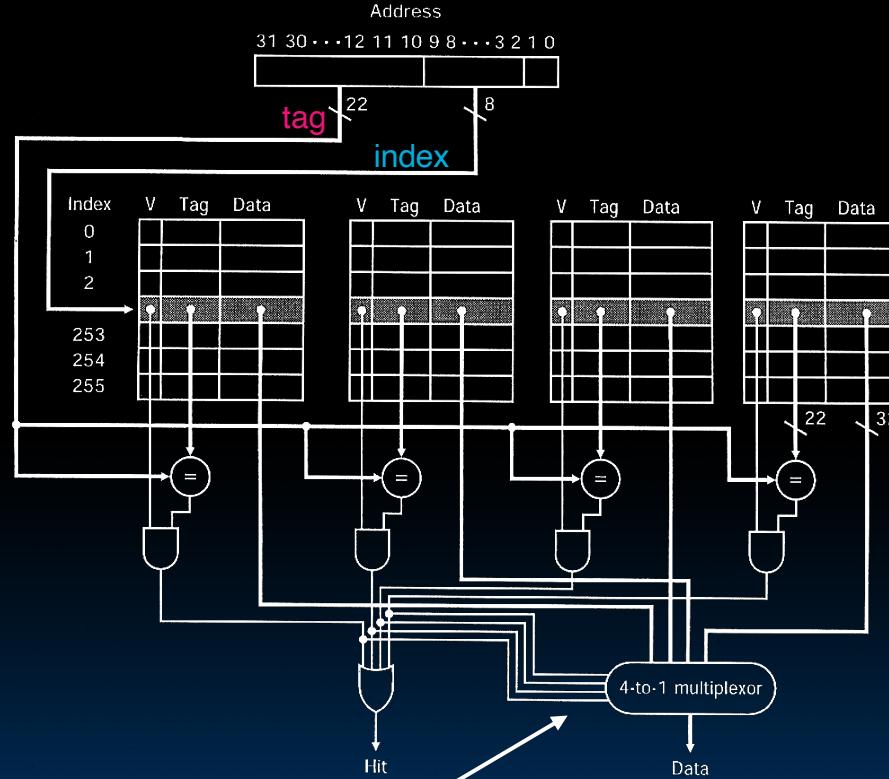
- Given memory address:

- Find correct set using Index value.
  - Compare Tag with all Tag values in that set.
  - If a match occurs, hit!, otherwise a miss.
  - Finally, use the offset field as usual to find the desired data within the block.

# N-Way Set Associative Cache (3/3)

- What's so great about this?
  - even a 2-way set assoc cache avoids a lot of conflict misses
  - hardware cost isn't that bad: only need  $N$  comparators
- In fact, for a cache with  $M$  blocks,
  - it's Direct-Mapped if it's 1-way set assoc
  - it's Fully Assoc if it's  $M$ -way set assoc
  - so these two are just special cases of the more general set associative design

# 4-Way Set Associative Cache Circuit



"One Hot" Encoding



# Block Replacement with Example

# Block Replacement Policy

- **Direct-Mapped Cache**
  - index completely specifies position which position a block can go in on a miss
- **N-Way Set Assoc**
  - index specifies a set, but block can occupy any position within the set on a miss
- **Fully Associative**
  - block can be written into any position
- **Question: if we have the choice, where should we write an incoming block?**
  - If there's a valid bit off, write new block into first invalid.
  - If all are valid, pick a **replacement policy**
    - rule for which block gets "cached out" on a miss.

# Block Replacement Policy

- LRU (Least Recently Used)
  - Idea: cache out block which has been accessed (read or write) least recently
  - Pro: temporal locality → recent past use implies likely future use: in fact, this is a very effective policy
  - Con: with 2-way set assoc, easy to keep track (one LRU bit); with 4-way or greater, requires complicated hardware and much time to keep track of this
- FIFO
  - Idea: ignores accesses, just tracks initial order
- Random
  - If low temporal locality of workload, works ok

# Block Replacement Example

- Our same 2-way set associative cache with a four byte total capacity and one byte blocks. We perform the following byte accesses:  
**0, 2, 0, 1, 4, 0, 2, 3, 5, 4**

- How many hits and how many misses will there be for the LRU replacement policy?



# Block Replacement Example: LRU

0: miss, bring into set 0 (loc 0)

2: miss, bring into set 0 (loc 1)

0: hit

1: miss, bring into set 1 (loc 0)

4: miss, bring into set 0 (loc 1, replace 2)

Addresses 0, 2, 0, 1, 4, 0, ...

0: hit

	loc 0	loc 1
set 0	0	lru
set 1		
set 0	lru	0
set 1		2
set 0	0	lru
set 1		2
set 0	0	lru
set 1	1	lru
set 0	lru	0
set 1	1	lru
set 0	lru	0
set 1	1	lru
set 0	0	lru
set 1	1	lru

# Cache Simulator!

[www.eecs.umass.edu/ece/koren/architecture/Cache/frame1.htm](http://www.eecs.umass.edu/ece/koren/architecture/Cache/frame1.htm)

**Block Replacement Simulator**

Cache Size  # Sets

LRU  
FIFO  
RAND Replacement Policy

Enter Query Sequence - Task A for Multi-tasking

in Decimal, or  Hex

```
0 2 0 1 4 0
```

Set Repeat  cycles

Task B (when multi-tasking)

**Cache Contents: LRU replacement policy;**

Set#	4 Block, 2-way set-associative cache - tags shown in red	
0	0 - 00000000	4 - 00000010
1	1 - 00000000	-

**COLOR KEY:** Compulsory Miss Capacity Miss Conflict Miss Cache Hit Unused

**Cache Query Results:**

Compulsory Misses :	3	Total Cache Queries :	6
Capacity Misses :	0	Total Misses :	4
Conflict Misses :	1	Miss Rate :	66.67 %
Cache Hits	2	Hit Rate :	33.33 %

**Cache Query Sequence Trace**

Cache Query Sequence Trace Address data replaced on miss shown in blue subscript

0	2	0	1	4 - 2	0
---	---	---	---	-------	---

Addresses 0, 2, 0, 1, 4, 0, ...

Should be compulsory

	loc 0	loc 1
set 0	0	Iru
set 1		
set 0	Iru	0
set 1		2
set 0	0	Iru
set 1		
set 0	0	Iru
set 1	1	Iru
set 0	Iru	0
set 1	1	Iru
set 0	Iru	0
set 1	1	Iru
set 0	0	Iru
set 1	1	Iru

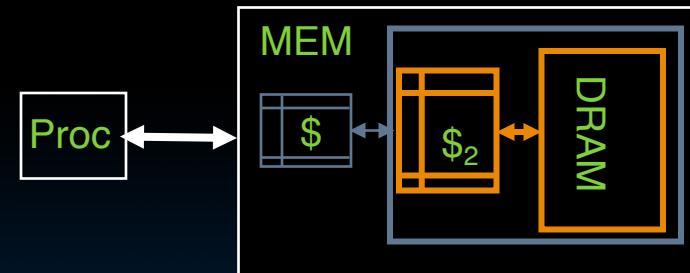


Average  
Memory Access  
Time (AMAT)

- How to choose between associativity, block size, replacement & write policy?
- Design against a performance model
  - Minimize: Average Memory Access Time
    - = Hit Time
    - + Miss Penalty × Miss Rate
  - influenced by technology & program behavior
- Create the illusion of a memory that is large, cheap, and fast - on average
- How can we improve miss penalty?

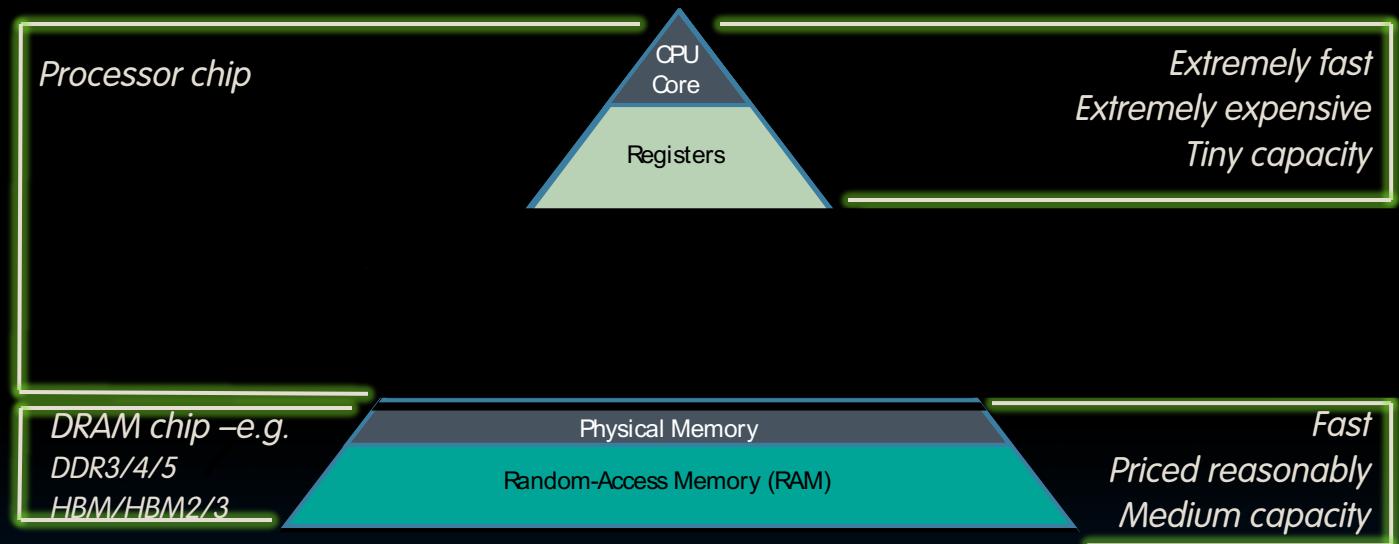
# Improving Miss Penalty

- When caches first became popular, Miss Penalty ~ 10 processor clock cycles
- Today 3 GHz Processor (1/3 ns per clock cycle) and 80 ns to go to DRAM  
~200 processor clock cycles!



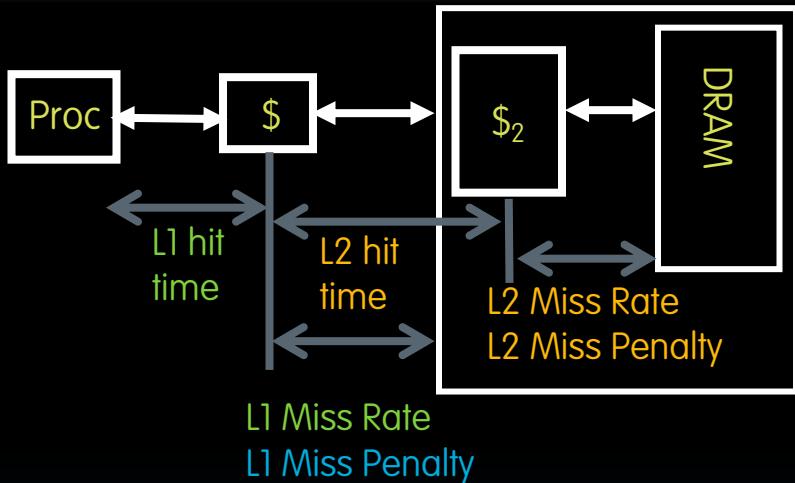
Solution: another cache between memory and the processor cache: Second Level (L2) Cache

# Great Idea #3: Principle of Locality / Memory Hierarchy



Let's see Cache configuration on Dan's computer...

# Analyzing Multi-level cache hierarchy



$$\text{Avg Mem Access Time} = \text{L1 Hit Time} + \text{L1 Miss Rate} * \text{L1 Miss Penalty}$$

$$\text{L1 Miss Penalty} = \text{L2 Hit Time} + \text{L2 Miss Rate} * \text{L2 Miss Penalty}$$

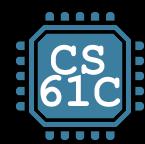
$$\text{Avg Mem Access Time} = \text{L1 Hit Time} + \text{L1 Miss Rate} * (\text{L2 Hit Time} + \text{L2 Miss Rate} * \text{L2 Miss Penalty})$$

# Example

- Assume
  - Hit Time = 1 cycle
  - Miss rate = 5%
  - Miss penalty = 20 cycles
  - Calculate AMAT...
- Avg mem access time
  - $= 1 + 0.05 \times 20$
  - $= 1 + 1$  cycles
  - $= 2$  cycles

# Ways to reduce miss rate

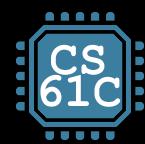
- Larger cache
  - limited by cost and technology
  - hit time of first level cache < cycle time (bigger caches are slower)
- More places in the cache to put each block of memory – associativity
  - fully-associative
    - any block any line
  - N-way set associated
    - N places for each block
    - direct map: N=1



# Typical Scale

---

- L1
  - size: tens of KB
  - hit time: complete in one clock cycle
  - miss rates: 1-5%
- L2:
  - size: hundreds of KB
  - hit time: few clock cycles
  - miss rates: 10-20%
- **L2 miss rate is fraction of L1 misses that also miss in L2**
  - why so high?



# Example: with L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L2 Hit Time = 5 cycles
  - L2 Miss rate = 15% (% L1 misses that miss)
  - L2 Miss Penalty = 200 cycles
- L1 miss penalty =  $5 + 0.15 * 200 = 35$
- Avg mem access time =  $1 + 0.05 \times 35$   
= 2.75 cycles

# Example: without L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L1 Miss Penalty = 200 cycles
- Avg mem access time =  $1 + 0.05 \times 200$   
= 11 cycles
- 4x faster with L2 cache! (2.75 vs. 11)



# Actual CPUs

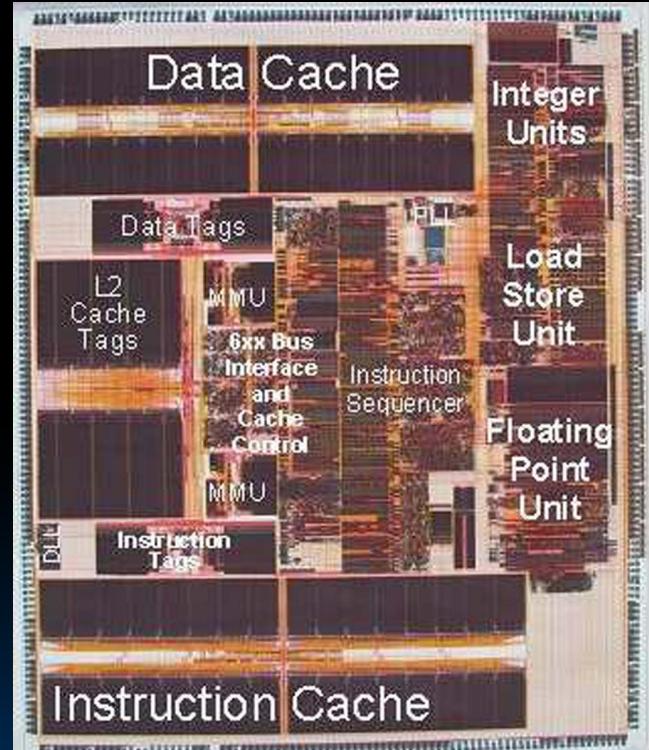
# An Actual CPU – Early PowerPC

- Cache

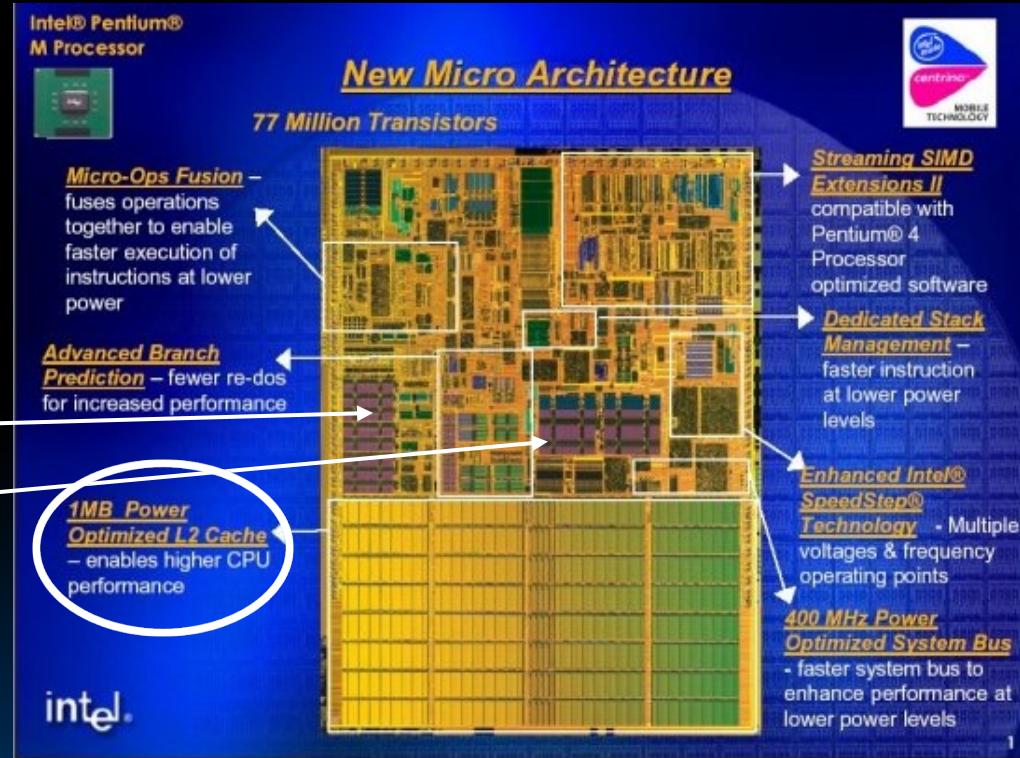
- 32 KiB Instructions & 32 KiB Data L1 caches
- External L2 Cache interface with integrated controller and cache tags, supports up to 1 MiB external L2 cache
- Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)

- Pipelining

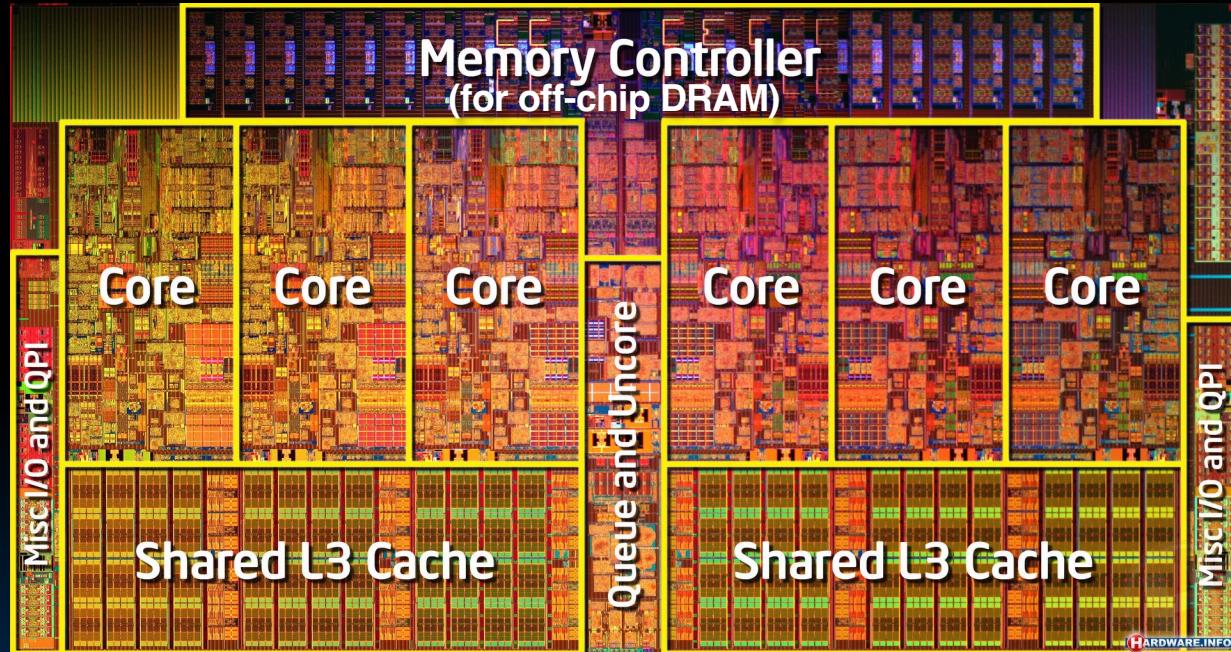
- Superscalar (3 inst/cycle)
- 6 execution units (2 integer and 1 double precision IEEE floating point)



# An Actual CPU – Pentium M



# An Actual CPU – Intel core i7



 When poll is active, respond at [pollev.com/ddg](http://pollev.com/ddg)

 Text **DDG** to **22333** once to join

L27 A 2-way set-associative cache cannot be outperformed by a direct-mapped cache.

The worst case for N-way set-associative cache is a repeated pattern of N addresses that map to that set.

On recent chips L4 cache is on chip.

FFF  
FFT  
FTF  
FTT  
TFF  
TFT  
TTF  
TTT

# How can DM beat 2-way?

- Sure, consider the caches from the previous slides (2-way set associative cache with a four-byte total capacity and one-byte blocks) with the following workload:

0, 2, 0, 4, 2

2-way: 0m, 2m, 0h, 4m, 2m;

DM: 0m, 2m, 0h, 4m, 2h

# And in Conclusion...

- We've discussed memory caching in detail. Caching in general shows up over and over in computer systems
  - Filesystem cache, Web page cache, Game databases / tablebases, Software memoization, Others?
- Big idea: if something is expensive but we want to do it repeatedly, do it once and cache the result.
- Cache design choices:
  - Size of cache: speed v. capacity
  - Block size (i.e., cache aspect ratio)
  - Write Policy (Write through v. write back)
  - Associativity choice of N (direct-mapped v. set v. fully associative)
  - Block replacement policy
  - 2nd level cache?
  - 3rd level cache?
- Use performance model to pick between choices, depending on programs, technology, budget, ...