Advanced Computer Architecture

COMP 5123

Fall 2016

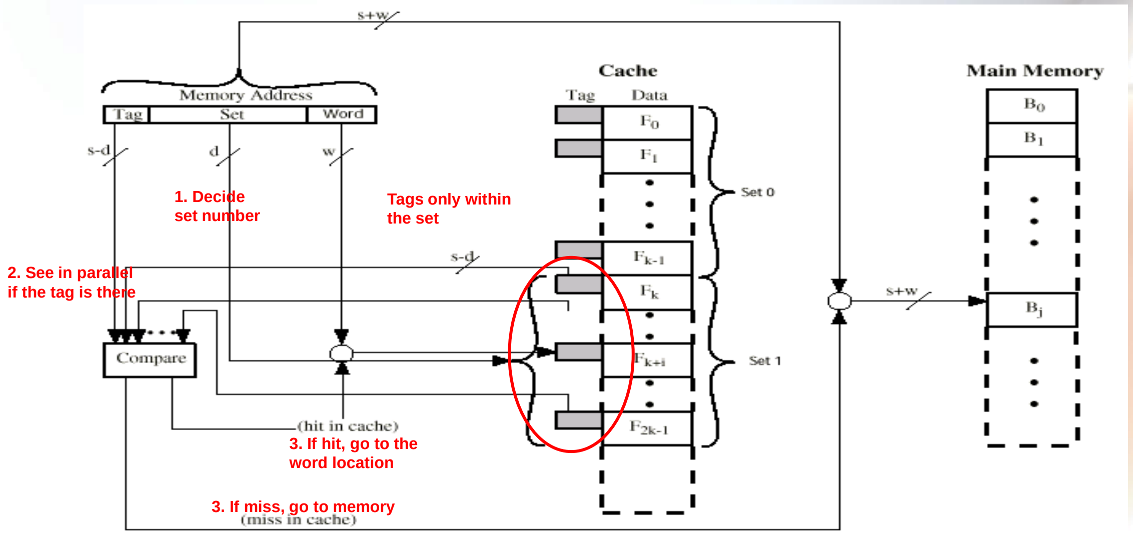
Computer Science Department

Prairie View A&M University

Mary Heejin Kim (aka Heejin Lim), Ph.D.

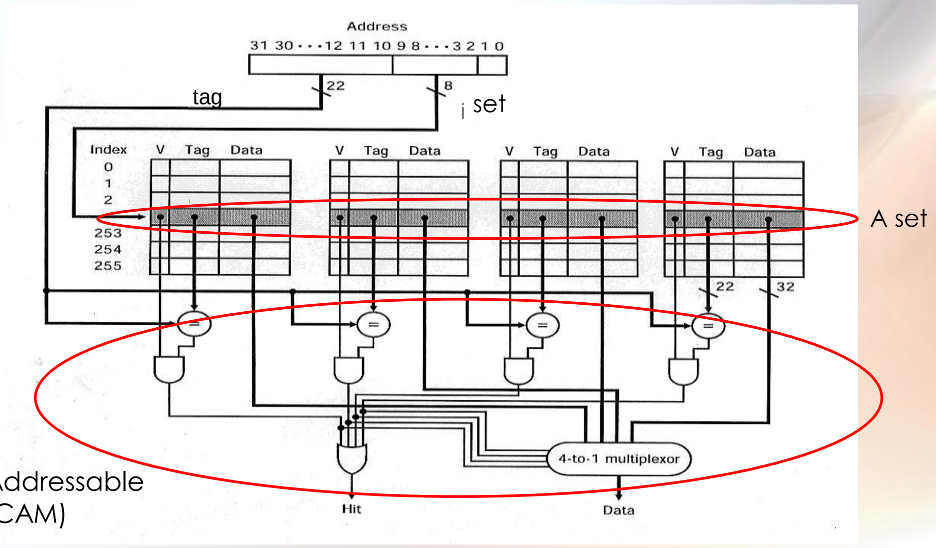
Chapter 4 Cache Memory (part 2)

K-Way Set Associative Cache Organization



3

4-Way Set Associative Cache Circuit



Content Addressable Memory (CAM)

4

k-Way Set Associative Cache

- Benefit
  - Better flexibility than direct mapped
  - Hardware cost isn't that bad: only need k comparators

Set Associative Mapping

- *v* sets, each with *k* lines
- m = total number of lines in the cache
- Extreme case 1
  - v=m, k=1 ➔ direct mapping
    - number of sets = number of total cache lines
    - 1 line per each set
- Extreme case 2
  - v=1, k=m ➔ fully associative mapping
    - only 1 big set
    - m lines for this big set
- What **k** is best?
  - 2~8 lines per set is most common

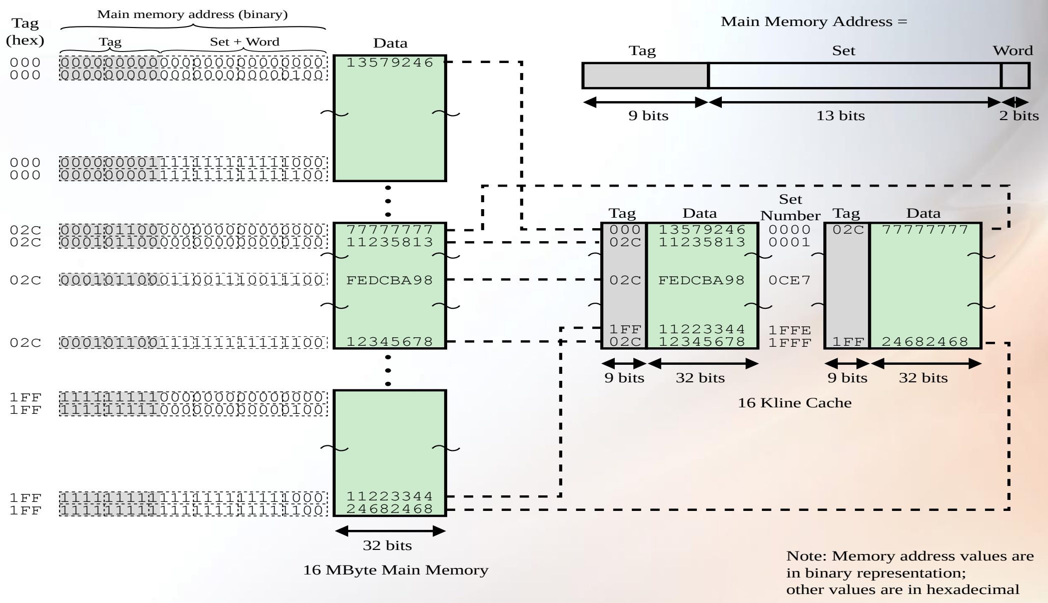
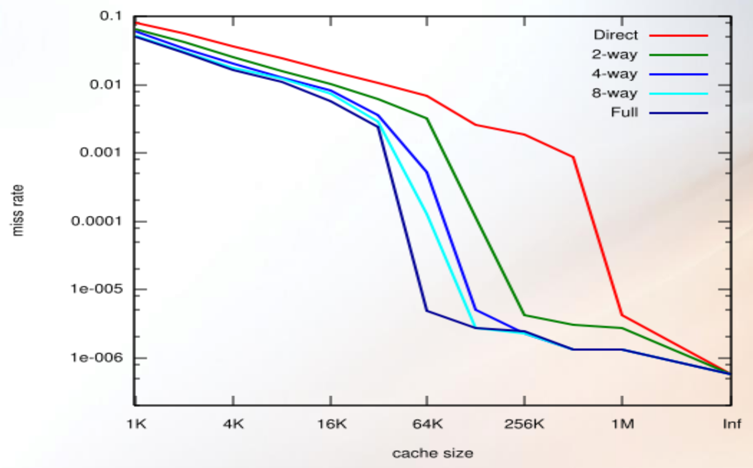


Figure 4.15 Two-Way Set Associative Mapping Example

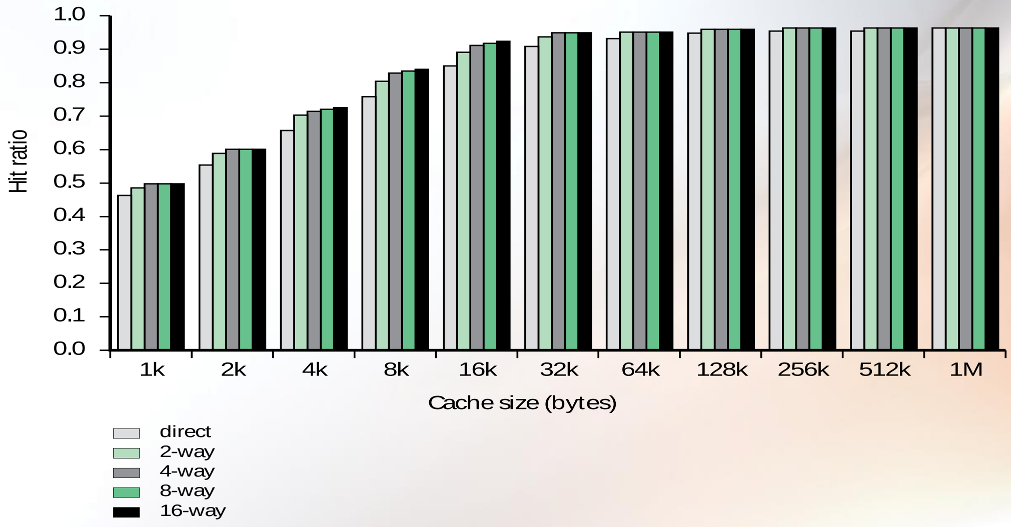
What k is best?



Cache Performance for SPEC CPU2000 Benchmarks (2003)

http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/

Varying Associativity over Cache Size



9

Set Associative Cache Performance

- Significant improvement up to at least 64kB for 2-way
- When the cache size is bigger, the k number doesn't play a great role
  - Difference between 2-way and 4-way at 4kB much less than 4kB to 8kB
  - Not justified against increasing cache to 8kB or 16kB
  - Above 32kB gives no improvement
- Cache complexity increases with associativity

10

Mapping Comparison

- Direct mapping
  - Tag, *line number*, word
  - **8 bit,** 14 bits, 2 bits
- Fully associative mapping
  - Tag, word
  - **22 bits**, 2 bits
- Set associative mapping (2-way, 2^{13} sets)
  - Tag, *set number*, word
  - **9 bits,** 13 bits, 2 bits
    - 2 lines per set: a memory block can go to either line
    - There are twice more candidate blocks for a particular set than in direct mapping → need one more bit for tag

Implicit

Other Cache Design Issues

Write Policy

When a block that is resident in the cache is to be replaced there are two cases to consider:

↓

If the old block in the cache has not been altered then it may be overwritten with a new block without first writing out the old block

↓

If at least one write operation has been performed on a word in that line of the cache then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block

There are two problems to contend with:

↓

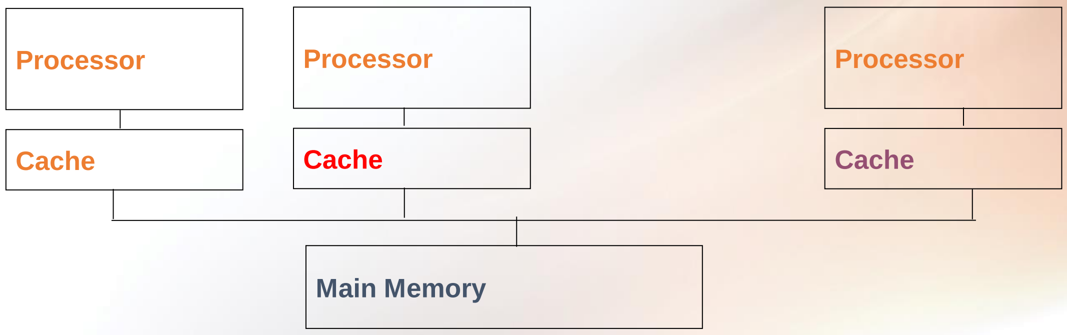
More than one device may have access to main memory

↓

A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache - if a word is altered in one cache it could conceivably invalidate a word in other caches

Cache Coherence problem

- More than one device have cache and MM is shared (e.g., multiprocessor system),
  - there is a difference between caches (even with write-through) **after cache write**



Cache Coherence problem

- Solutions
  - Bus watching with **write through**
    - All cache controllers use write through policy, and when write-through is detected, **invalidates its cache line**
  - Hardware transparency
    - **Updates** are made to **all caches** as well as to MM
  - Noncacheable memory
    - **Do not cache the shared memory** area (treated as cache miss)

15

Optimum Line Size

- Note: In Cache, it is a ***block*** that is moved, not an individual word
  - Adjacent words are retrieved, too
- Thanks to locality, **hit ratio increases** as cache line size (or MM block size) increases to some degree
- But when the block size becomes too large, cache miss increases. Why?

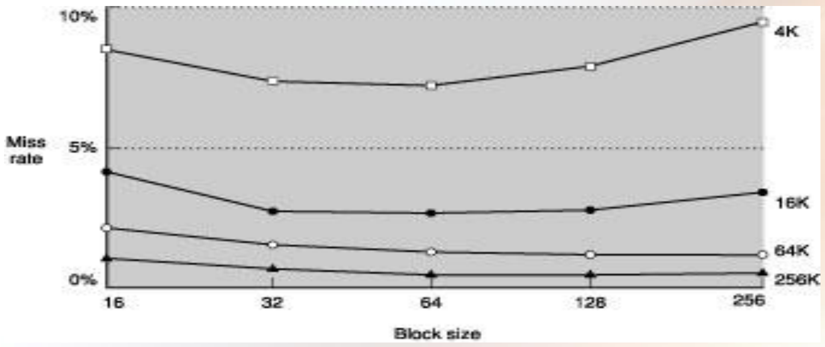
1. **Too big** block size → contains unneeded data→ **not much benefit from locality**
   - Thus unneeded data is brought in to cache, pushing out actually needed data.
2. Decreased number of blocks→ making the competition greater.
   - A block may be preempted more frequently (**higher miss rate**).

- And don't forget the **larger transfer time** with larger block size

16

Optimum line size

- Not straightforward to find the optimum size
  - **8~64 bytes** are reasonable
  - 64~128 bytes for HPC



Causes of cache miss

1. **Compulsory**
   - Empty cache, cold boot, first miss
2. **Capacity**
   - The cache is not big enough to store the working set
3. **Conflict (or collision)**
   - In direct mapping or set associative mapping
   - The place is already occupied by other block(s)

	Direct mapping	8-way set associative
Compulsory	5 %	6 %
Capacity	73 %	94 %
Conflict	22 %	0 %

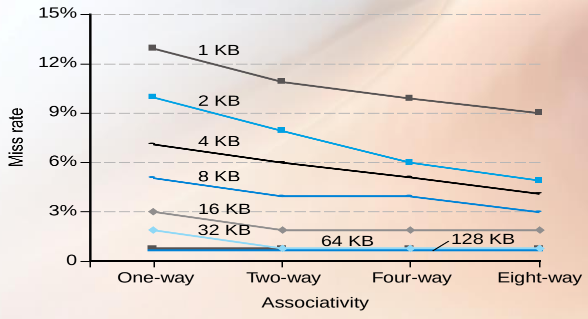
Small cache

	Direct mapping	8-way set associative
Compulsory	34 %	59 %
Capacity	16 %	27 %
Conflict	50 %	14 %

Large cache

Reducing cache miss

- Conflict misses can be reduced by **higher associativity**
  - But it **slows** access time, leading to lower overall performance
- Capacity misses can be reduced by **larger cache size**
  - But it **slows** access time
- The balance?
  - L1 cache → keep small
  - L2 cache → larger



19

Number of caches

- Single Cache
  - First generation
- Multiple Caches
  1. Multilevel Caches
  2. Unified vs. split caches

20

1. Multilevel cache

- Growing CPU & Memory speed gap
  - When caches first became popular, Miss Penalty ~ 10 processor clock cycles
  - Today? Consider 2.5 GHz Processor (0.4 ns per clock cycle) with 80 ns to access DRAM ➔ 200 clock cycles!
- Solution? Multilevel cache!
  - Fast primary cache and slower secondary cache

Does multilevel cache help?

- Consider a processor
  - Clock rate = 5 GHz (each clock cycle = 0.2 ns)
  - CPI = 1, if no cache miss
  - Main memory access time = 100 ns
  - Primary cache miss rate = 2%
- With single cache at CPU speed
  - The miss penalty = 100 ns = 500 clock cycles
  - Effective CPI = 1 * 98% + (1 + 500) * 2%
    = 0.98 + 10.02 = **11**
- Now, add a secondary cache with 5 ns access time and miss rate of 0.5%
  - The miss penalty = 5 ns = 25 clock cycles
  - Effective CPI = 1 * 98% + (1 + 25) * 1.5%
    + (1+ 25 + 500) * 0.5% = 0.98 + 0.39 + 2.63 = **4.0**
- **CPI of 11.0 became 4.0!**

Cache design parameters example

Feature	L1 cache	L2 cache
Total size in **Kbytes**	16 - 64	500 - 8000
Block size in **bytes**	32 - 64	32 - 128
Miss penalty in **clocks**	10 - 25	100 - 1000
Miss rate	**2% - 5%**	**0.1% - 2%**

23

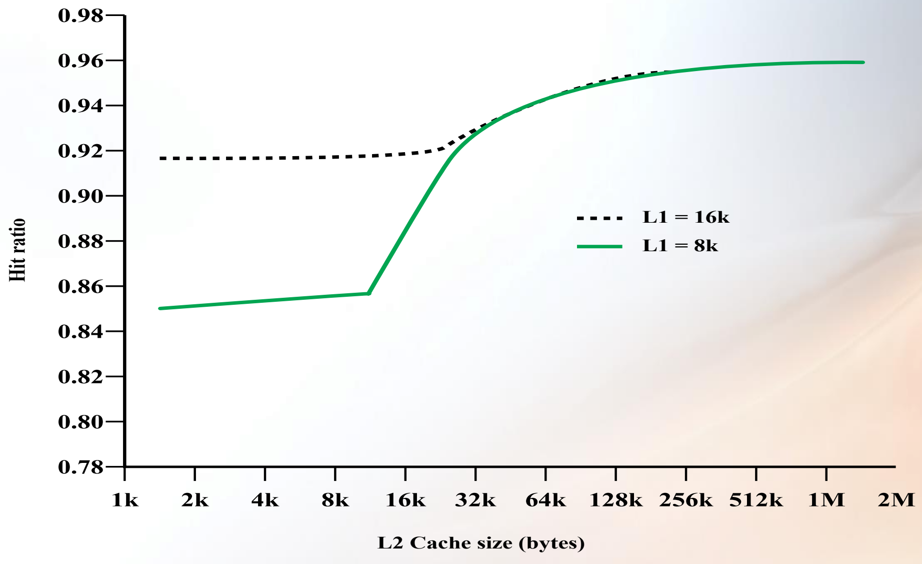


Figure 4.17 Total Hit Ratio (L1 and L2) for 8 Kbyte and 16 Kbyte L1

Multilevel Cache implementation

- Primary cache (high speed)
  - Faster access time and smaller
  - **Smaller associativity** - faster access
  - **Smaller block size** - faster transfer time
- Secondary cache (low miss rate)
  - Slower but larger –reduces capacity miss
  - Larger associativity (4 or higher) –reduces conflict miss
  - Larger block size – includes enough data for locality. With large number of blocks, conflict miss is low
- L3 or off-chip caches (economy)
  - Even slower and larger
  - Low associativity (2 or less) – due to large size, no difference by associativity. Just avoid complexity

25

2. Unified vs. Split Cache

- MM content = Instruction + Data
  - Shall we store them in a single cache (Unified) or separate caches (Split cache)?
- Advantages of **unified cache**
  - Automatic balancing: if one of them uses more cache, there is no hard boundary
  - Simpler design and implementation
- Advantages of **Split Cache**
  - Instruction Fetch → Decode → Operand Fetch → Execution
  - No contention between instruction fetch and operand fetch in **pipelining**

26

Harvard Architecture - FYI

- A computer architecture with physically separate storage and signal pathways for instructions and data.
  - The term originated from the Harvard Mark I computer (1944)
  - Word width may differ, instruction memory can be read-only, etc.
- Von Neumann Architecture
  - Uses a single storage to hold both instruction and data
  - Both cannot be read at the same time

27

Review - Comparison of Cache Sizes

Instr./data cache (Harvard arch)

Processor	Type	Year of Introduction	L1 cache[a]	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/server	2002	32 KB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 KB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 KB/64 KB	1MB	—

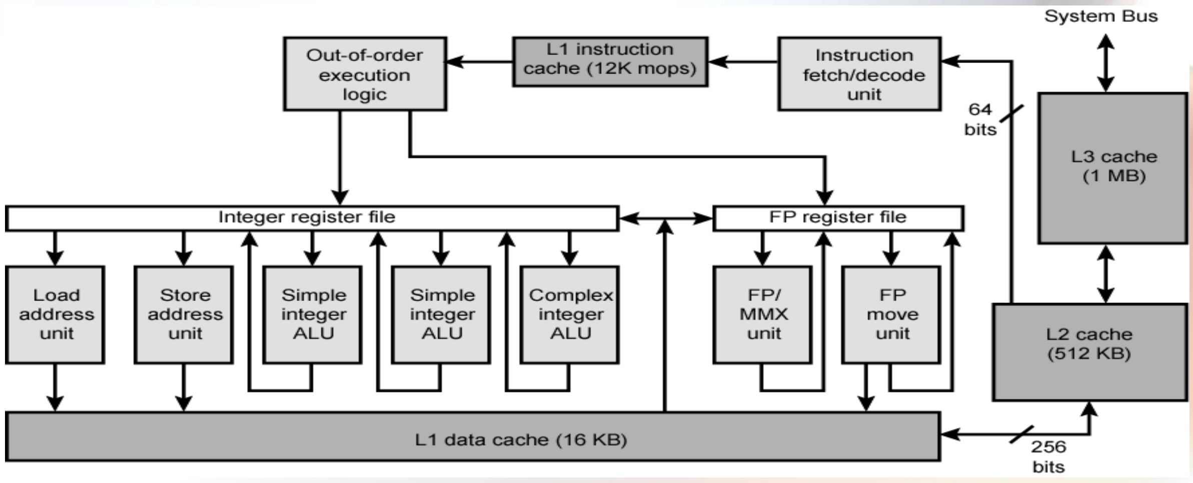
28

Pentium 4 Cache

- 80486 – 8KB using 16-byte lines and 4-way set associative cache
- Pentium (all versions) – two on-chip L1 caches
  - Data & instructions
- Pentium III – L3 cache added off chip
- Pentium 4 (original in 2000)
  - L1 caches
    - 8KB data, 12 KB instruction
    - 64-byte lines
    - 4-way set associative
  - L2 cache
    - Feeding both L1 caches
    - 256 KB
    - 128-byte lines
    - 8-way set associative
  - L3 cache on chip (with Extreme edition)
- Intel Core 2 Extreme
  - 24-way associativity L2 cache

29

Pentium 4 Block Diagram



(cache sizes – different by version)

30

Table 4.5 Pentium 4 Cache Operating Modes

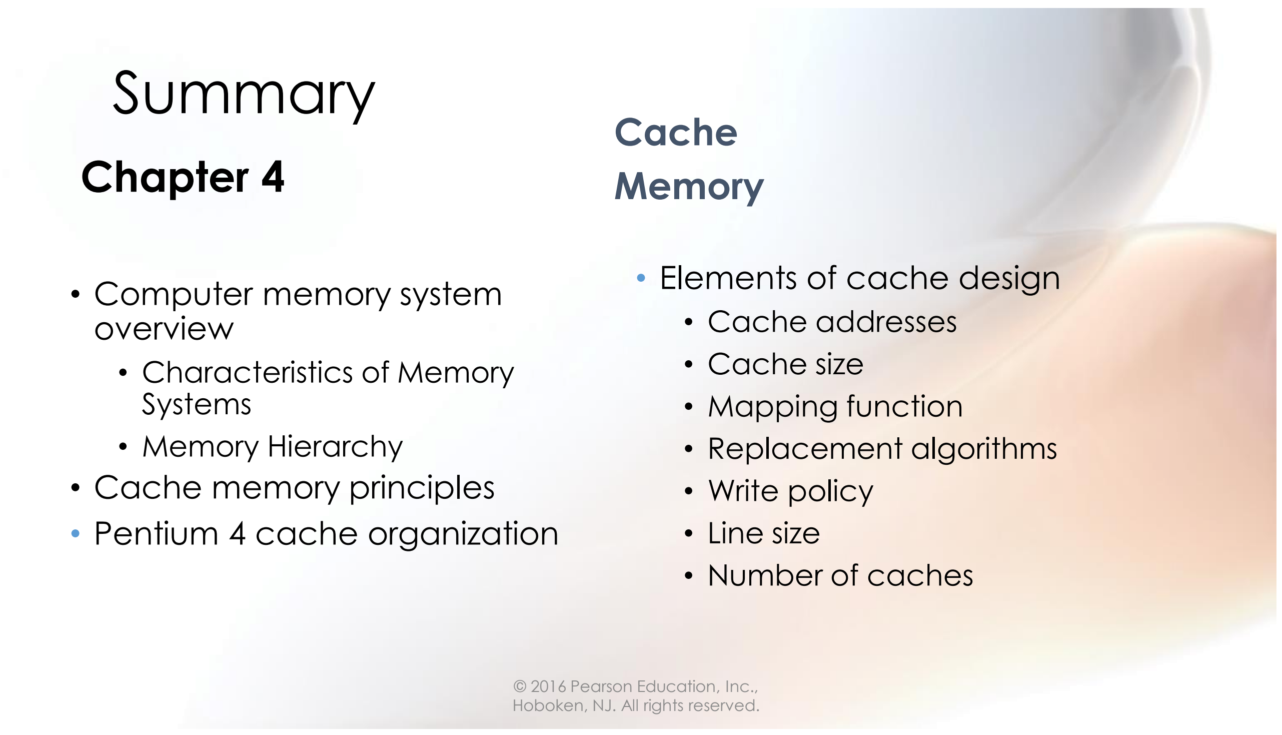
Control Bits		Operating Mode		
CD	**NW**	**Cache Fills**	**Write Throughs**	**Invalidates**
0	0	Enabled	Enabled	Enabled
1	0	Disabled	Enabled	Enabled
1	1	Disabled	Disabled	Disabled

Note: CD = 0; NW = 1 is an invalid combination.

Pentium 4 Design Reasoning

- **Instruction cache** is between **instruction decode logic** and **execution core**. Why?
  - Decodes instructions into RISC-like micro-ops (mops) before L1 cache
    - Micro-ops = fixed length
      - Superscalar pipelining and scheduling
  - Pentium instructions are long & complex
  - Performance improved by separating decoding from scheduling & pipelining
    - (More later – ch14)
- Data cache is **write back**
  - Can be configured to **write through**
- Both L2 and L3
  - 8-way set-associative
  - Line size = 128 bytes

32



Summary

Chapter 4

Cache Memory

- Computer memory system overview
  - Characteristics of Memory Systems
  - Memory Hierarchy
- Cache memory principles
- Pentium 4 cache organization

- Elements of cache design
  - Cache addresses
  - Cache size
  - Mapping function
  - Replacement algorithms
  - Write policy
  - Line size
  - Number of caches