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

Memory System Characteristics

Location Internal (e.g. processor registers, cache, main memory) External (e.g. optical disks, magnetic disks, tapes) **Capacity** Number of words Number of bytes **Unit of Transfer** Word Block **Access Method** Sequential Direct Random Associative	**Performance** Access time Cycle time Transfer rate **Physical Type** Semiconductor Magnetic Optical Magneto-optical **Physical Characteristics** Volatile/nonvolatile Erasable/nonerasable **Organization** Memory modules

3

1. Where are the memories?

- CPU
  - Register
  - Control Unit – own memory
- Internal
  - Cache
  - RAM
- External
  - Disk
  - Tape

4

2. Capacity

- Measured in *Bytes* or *Words*
- Word size
  - The natural unit of organization
  - 8, 16, 32 bits
  - Typically
    - Word size = integer size (`int` in C = word size)
    - Word size = instruction size
- Addressable unit
  - Smallest location which can be uniquely addressed
  - Mostly bytes, but may be word

5

3. Unit of Transfer

- Internal
  - Usually governed by **data bus width**
  - Equal to the number of electrical lines into and out of the memory module
  - Could be same size as word or larger (64, 128, 256 bits)
- External
  - Usually a **block** which is much larger than a word

6

4. Access Methods (1)

1. **Sequential**
   - Start at the beginning and read through in order
   - Access time depends on location of data and previous location
   - Uses shared read/write mechanism (i.e., only one location can be read at one time)
   - e.g. Tape
2. **Direct**
   - Shared read/write mechanism
   - Individual ***blocks*** or ***records*** have unique address
   - Access is by **jumping to vicinity** + sequential search, counting, waiting, etc.
   - **Access time** depends on **location** and previous location
   - e.g. Disk

7

4. Access Methods (2)

3. **Random**
   - Dedicated read/write mechanism (**not shared**) for each addressable location
   - Access time is **independent of location** or previous access = constant
   - e.g. Main memory, Cache
4. **Associative**
   - Data is located by a comparison with contents of a portion of the store (Content-Addressable Memory, CAM)
   - Access time is **independent of location** or previous access
   - e.g. Cache

8

5. Memory Performance

- **Access time (latency)**
  - Time between presenting the address and getting the valid data
  - For non-random access memory, time to position the R/W head to the desired location
- **Memory Cycle time**
  - Cycle time is ***access time + recovery***
  - Time may be required for the memory to "recover" before next access
    - Waiting for the transient signal to die out
    - Regeneration of the destroyed content
    - Concerned with the system bus, not the processor
- **Transfer Rate**
  - Rate at which data can be moved (bps)
  - For random access memory,
    = 1/(cycle time) bps

9

6. Physical Type, Characteristics

- Types
  - Semiconductor
    - Either volatile (RAM) or non-volatile (ROM)
  - Magnetic surface (disk, tape)
  - Optical (CD-ROM, DVD)
  - Other (Flash, Bubble, Hologram)
- Characteristics
  - Volatile vs. Non-volatile
    - Volatile memory: Information is lost at power-off
  - Erasable vs. Non-erasable
    - Non-erasable memory cannot be altered (ROM, CD-ROM)

10

Memory Selection

- Want a fast machine?
  - It is possible to build a computer which uses only static RAM (SRAM)
    - This would be very fast, and need no cache
    - This would cost a lot of **$$$$$$!**
- Trade-off across different memory technologies
  - Faster, more expensive
  - Larger, less expensive, but slower
- Need to consider all
  - Capacity (How much?)
  - Access time (How fast?)
  - Cost (How expensive?)
- How can you solve the dilemma?

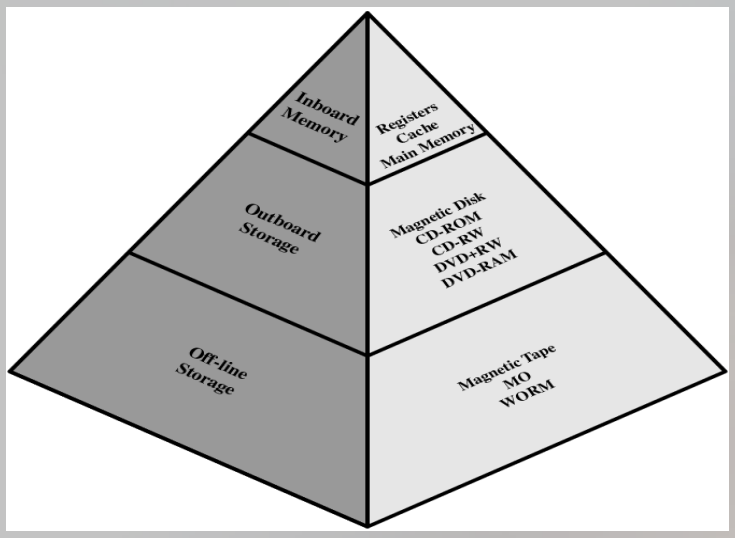
11

Hint - Locality of Reference

- **Locality**
  - If it rains at PVAMU campus, it may rain at the Houston (spatial)
  - If it rained at PVAMU campus at 12:30PM, it may still rain at 12:45PM (temporal)
- During the course of the execution of a program, **memory references tend to cluster**
  - For both data and instructions
    - e.g. loops, subroutines, tables, arrays
  - Spatial locality
  - Temporal Locality
    - Processor tends to access memory locations that have been used recently
    - It will change over long time, but **will be there for a short while**

12

Memory Hierarchy - Diagram



- Cheaper
- Larger
- Slower
- ***Less frequently used by processor***

13

2-level memory example

- **Faster** memory (smaller, expensive)
  - Currently accessed cluster
  - If the data/instruction is found, it is called ***hit***. Otherwise, ***miss***
- **Slower** memory (larger, cheap)
  - Contains whole data, program

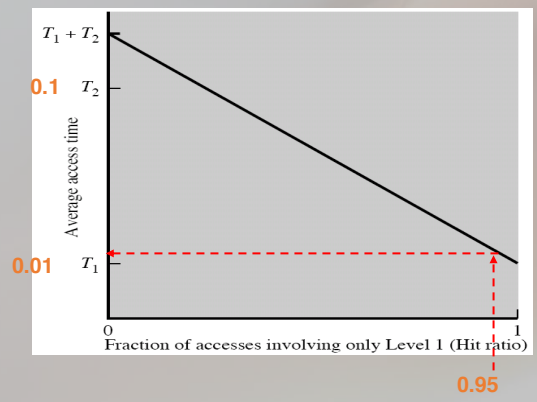
14

Performance improvement with 2-level memory

- Level 1: 1000 words with **0.01us (T_1)**
- Level 2: 100,000 words with **0.1us (T_2)**

- 95% time → found in Level 1
- 5% time → need to access Level 2
  - Access time for 95% time = **0.95 * 0.01 us = 0.0095**
  - Access time for 5% time = **0.05 * (0.01 us + 0.1 us) = 0.0055**

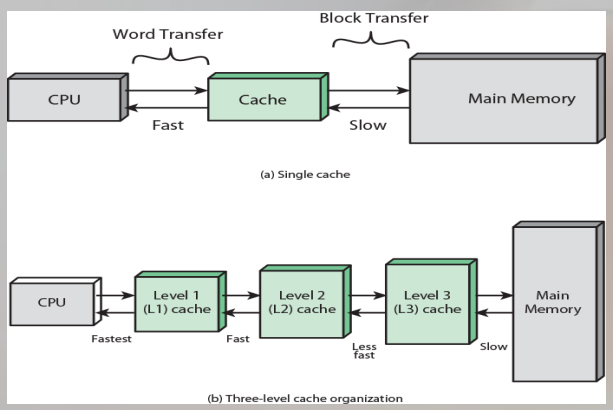
- then average access time = **0.0095 + 0.0055 = 0.015 us**
  Not bad!



15

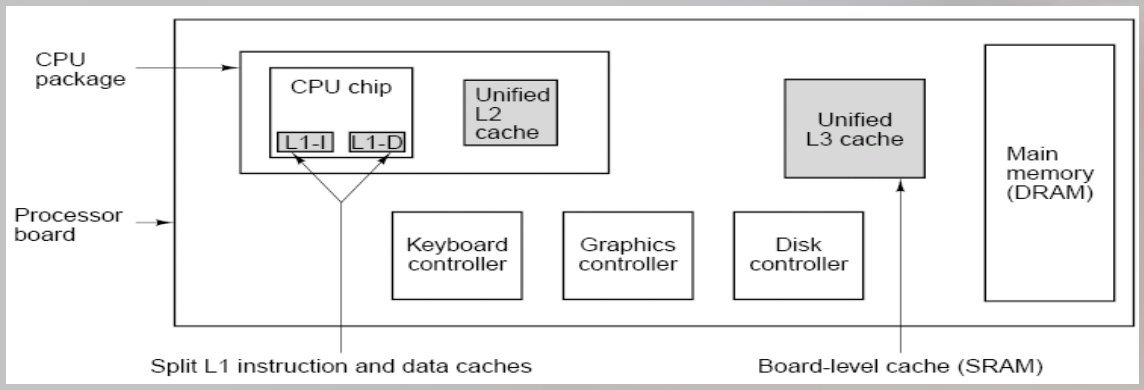
Faster memory → Cache

- Small amount of fast memory
  - SRAM (flip-flop), similar logic as processor, power-hungry
- Sits between normal main memory and CPU
- May be located on CPU chip or module



16

Location of Cache



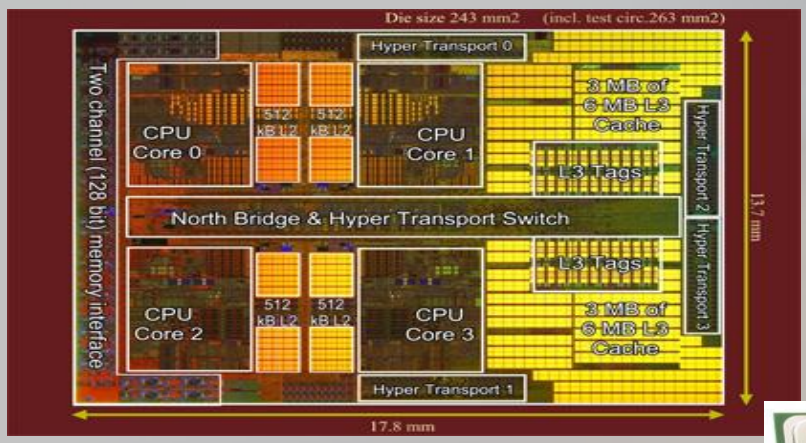
17

The Differences

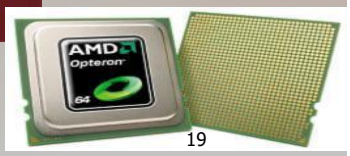
- **L1 cache**
  - Operates at register speed (e.g. 0.3 ns for 3GHz speed)
  - Memory accesses within a single processor cycle
- **L2 cache**
  - SRAM
  - Accessed within two to four processor cycle
- **L3 cache**
  - May reside on the motherboard (some have on-chip)
  - 8-10 ns
- **Main memory**
  - DRAM: around 50-60 nanoseconds
  - SDRAM: 2-10 ns after first slow access
- **Hard disk**
  - Mechanical, slow

18

On chip cache memory

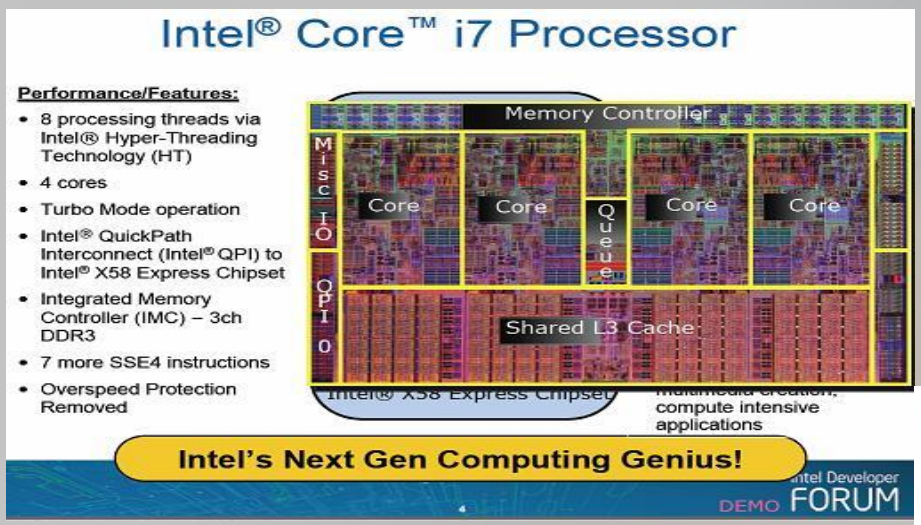


AMD Quadcore Opteron (Shanghai) chip (2008)



19

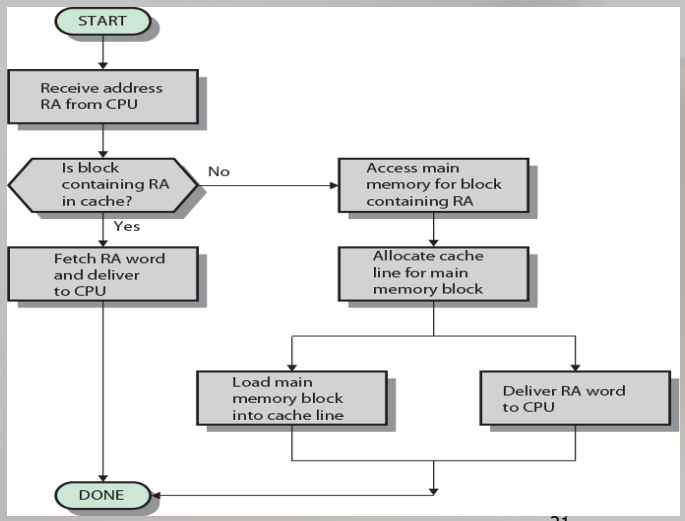
On chip cache memory



20

Cache Read Operation

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU (or do it in parallel)

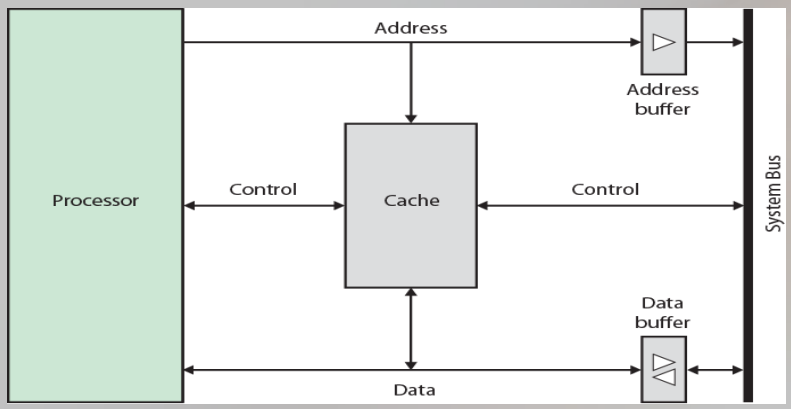


RA: Read Address

21

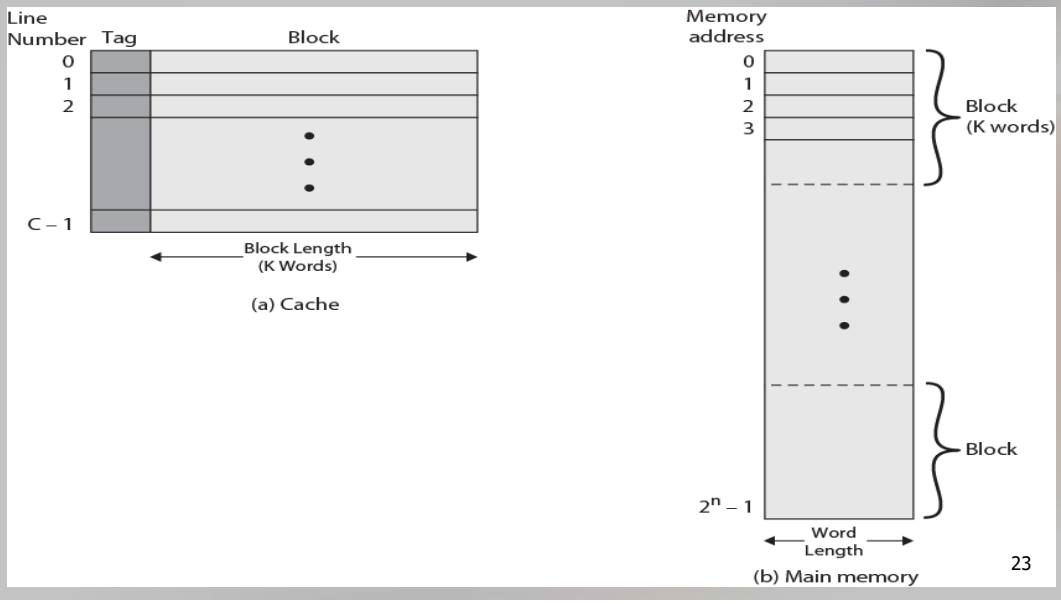
Typical Cache Organization

- When cache hit occurs, data and address buffers are disabled, communication is only btw processor and cache.



22

Cache/Main Memory Structure



23

Cache Design - Overview

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	Write once
Mapping Function	**Line Size**
Direct	**Number of caches**
Associative	Single or two level
Set Associative	Unified or split
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

24

Cache Addressing (1)

- Virtual memory
  - allows program to address memory from a logical point of view without regard to the amount of main memory physically available (chap 8).
  - memory management unit (MMU) translates each virtual address into a physical address in main memory
- Where does cache sit?
  - Between processor and MMU (logical cache)
  - Between MMU and main memory (physical cache)

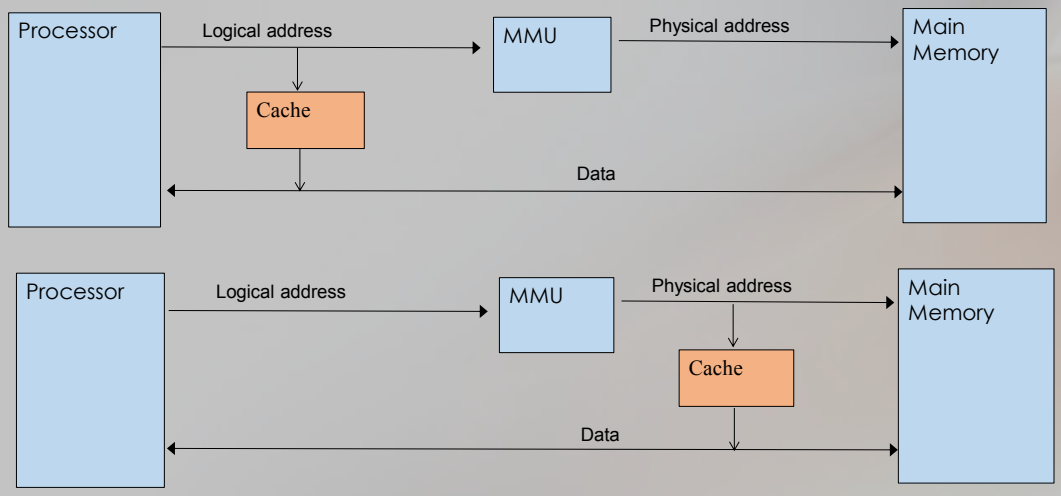
25

Cache Addressing (2)

- **Logical cache** (virtual cache) stores data using virtual addresses
  - Processor accesses cache directly, not through MMU
  - Cache access **faster**, before MMU address translation
  - Same virtual address in two different applications refers to two different physical address
    - Must flush cache on each context switch, or
    - Extra bit to identify
- **Physical cache** stores data using main memory physical addresses

26

Logical cache vs. physical cache



27

Size does matter

- Why small is better?
  - More cache is expensive
  - Use as little as possible
- Why larger is better?
  - More cache is faster thanks to higher hit ratio (up to a point)
    - Larger cache is slightly slower than smaller cache due to number of gates
  - Use as much as possible
- So where is the "optimum" size?

28

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	—

29

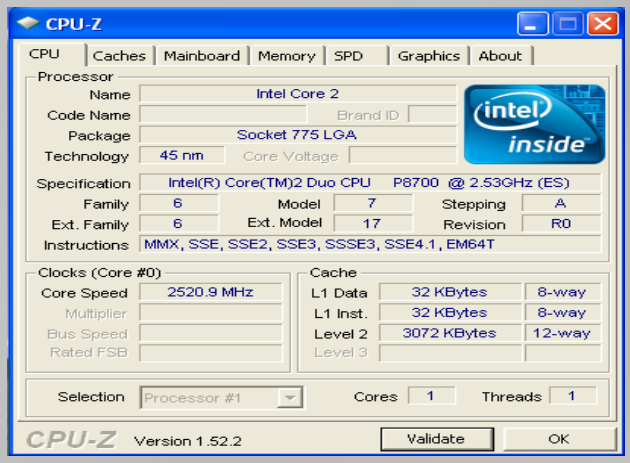
Latest Cache size

- P4 Extreme Edition (2003)
  - **8 KB L1 data + 12 KB L1 instruction**
  - 512 KB L2
  - 2 MB L3
- Pentium Extreme Edition (2005)
  - 16 KB L1 data + 12 KB L1 instruction *Per Core*
  - 2 MB L2 *Per Core*
- Intel Core 2 (2006)
  - 64KB L1 per core
  - 4 MB L2
- Intel Core i7 (2008, 64-bit)
  - 256KB L2
  - 8MB L3
- IBM Power 4 series has a 256MB L3 cache shared among several processors
  - >$1000

30

Cache information on your PC

- CPUID program
  - Download from http://www.cpuid.com/cpuz.php



Cache Mapping

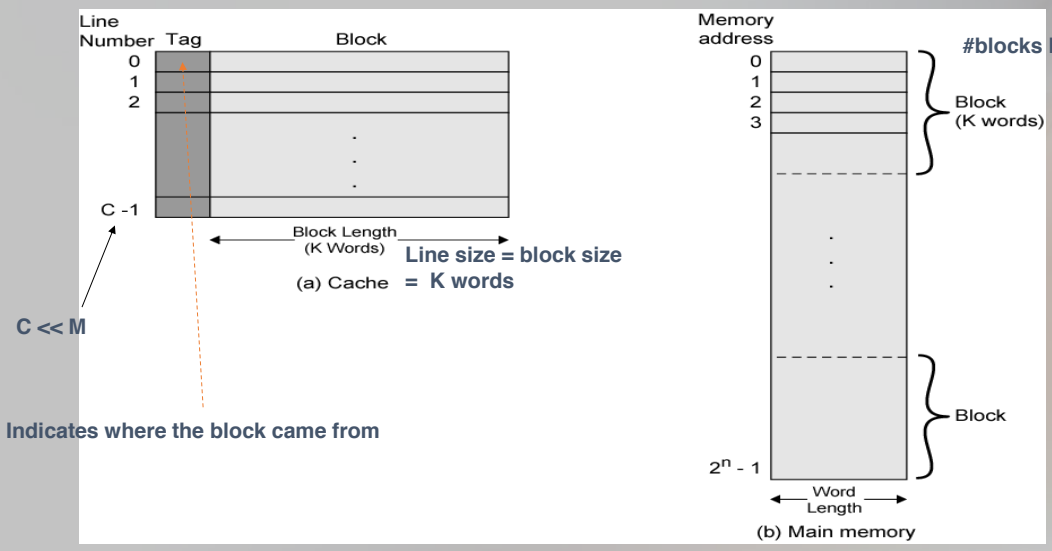
Mapping Function

- Cache lines << memory blocks
  - Need to map memory block to cache
  - Mapping algorithm
  - Which memory block is in cache?

- 3 methods
  1. Direct
  2. Associative
  3. Set associative

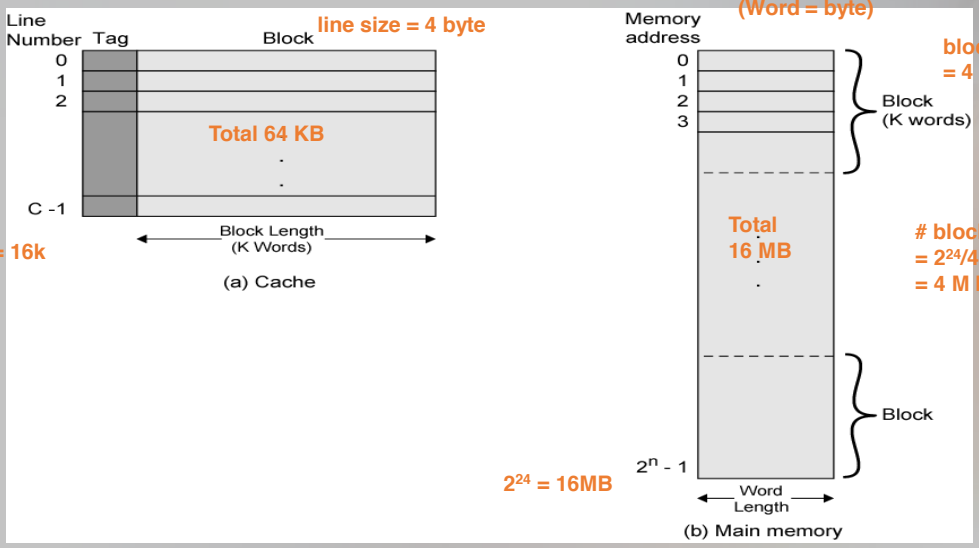
33

Cache/Main Memory Structure



34

Example to be used



lines
$C = 2^{14} = 16k$

$2^{24} = 16MB$

blocks
$= 2^{24}/4 = 2^{22}$
$= 4$ M blocks

Cache mapping

1. Direct mapped
   - Only one place to go in cache
2. Fully associative
   - Anywhere to go in cache
3. Set associative
   - A memory block is mapped to a set
   - Within a set, it can be assigned to anywhere
   - If the number of elements in a set is *n*, it is called *n*-way

Cache Mapping

1. Direct Mapping

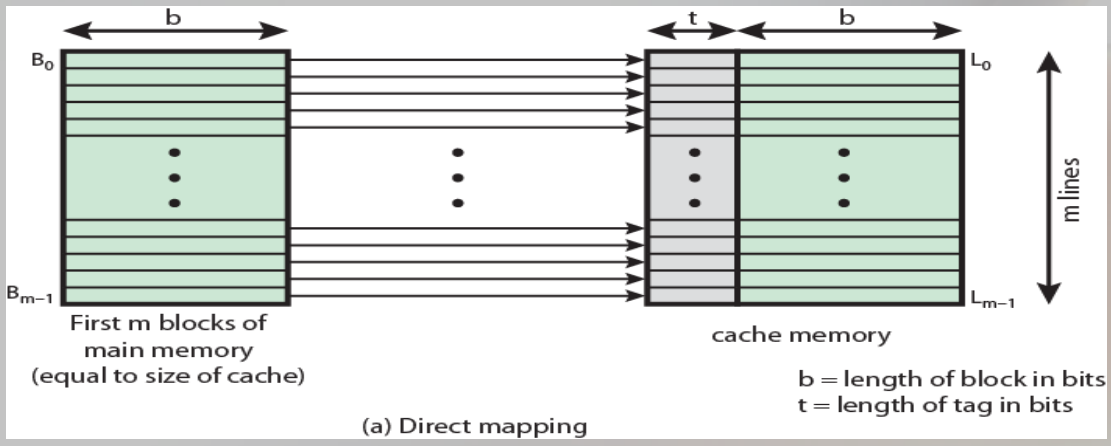
37

1. Direct Mapping (The simplest)

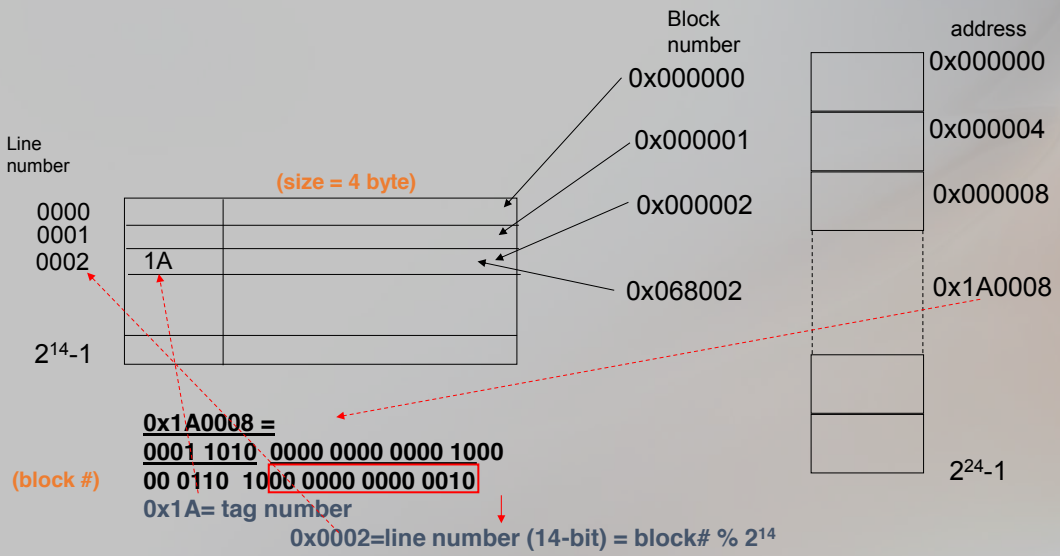
- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- ***i = j modulo m***
  - *i* = cache line number
  - *j* = main memory block number
  - *m* = number of lines in the cache
- Example
  - If there are **16 lines in the cache**, the memory block 21 is assigned to cache line **5 (= 21 % 16)**

38

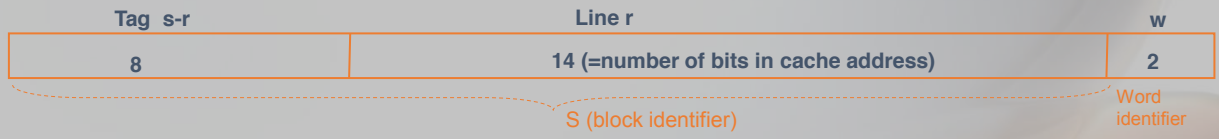
Main memory to Cache



Direct mapping example



Direct Mapping Address Structure

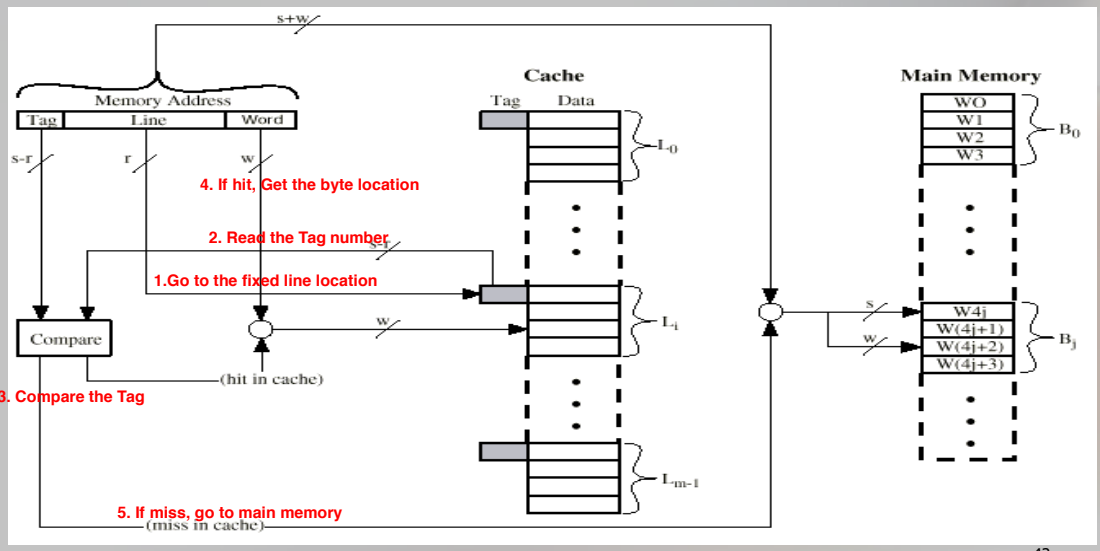


- **Address length** = (s + w) bits = 22 + 2 = 24 bits
- **Block size = line size** = 2^w bytes = 2^2 = 4 bytes
- **Number of blocks** in main memory
  = $2^{s+w} / 2^w = 2^s$ = 2^{22} blocks
- **Number of lines** in cache
  = m = 2^r = 2^{14} cache lines
- Cache line — Main Memory blocks held

Cache line	Main Memory blocks held
0	0, m, 2m, …, 2^s-m
1	1, m+1, 2m+1,…, 2^s-m+1
….	….
m-1	m-1, 2m-1, 3m-1,…, 2^s-1

41

Direct Mapping Cache Organization



42

Direct Mapping pros & cons

- **Good ☺**
  - Fixed location for given block
  - Simple & inexpensive
- **Bad ☹**
  - Fixed location for given block
  - Competition for the same line
    - If a program accesses 2 blocks that map to the **same line repeatedly**, they get swapped continually
      →**high cache miss**
    - Inefficient usage: cannot use other unused cache lines

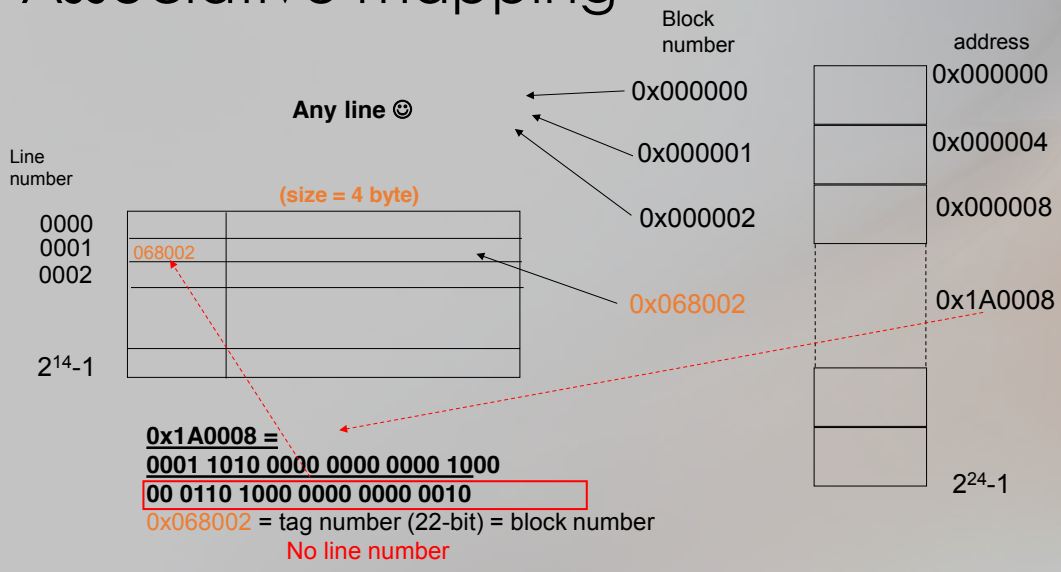
43

A small trick – Victim cache

- Lowers cache miss penalty
- Discarded cache data is already fetched, so it can be used again quickly. (no memory access)
  - But where do we store it?
- Victim cache
  - A fully associative cache
  - 4 to 16 cache lines
  - Between direct mapped L1 cache and next level of memory

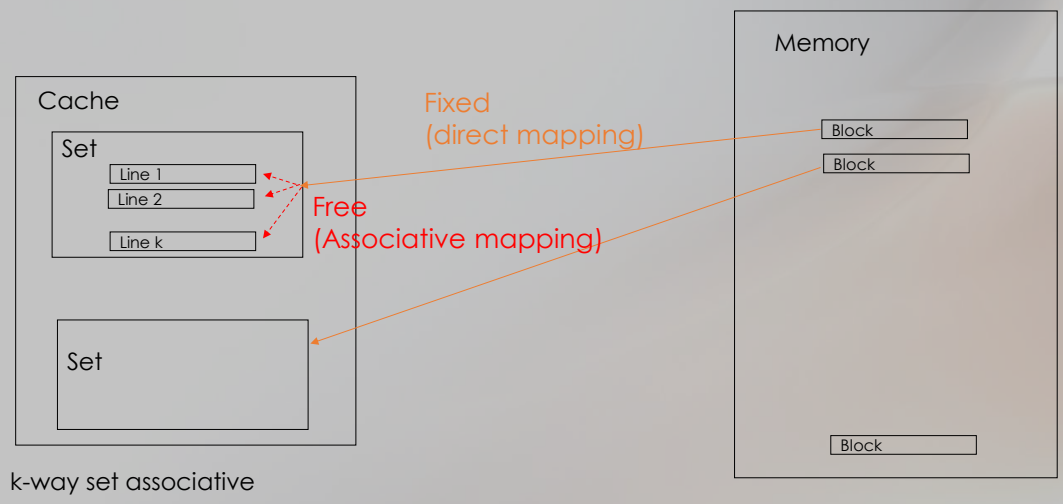
44

Associative mapping



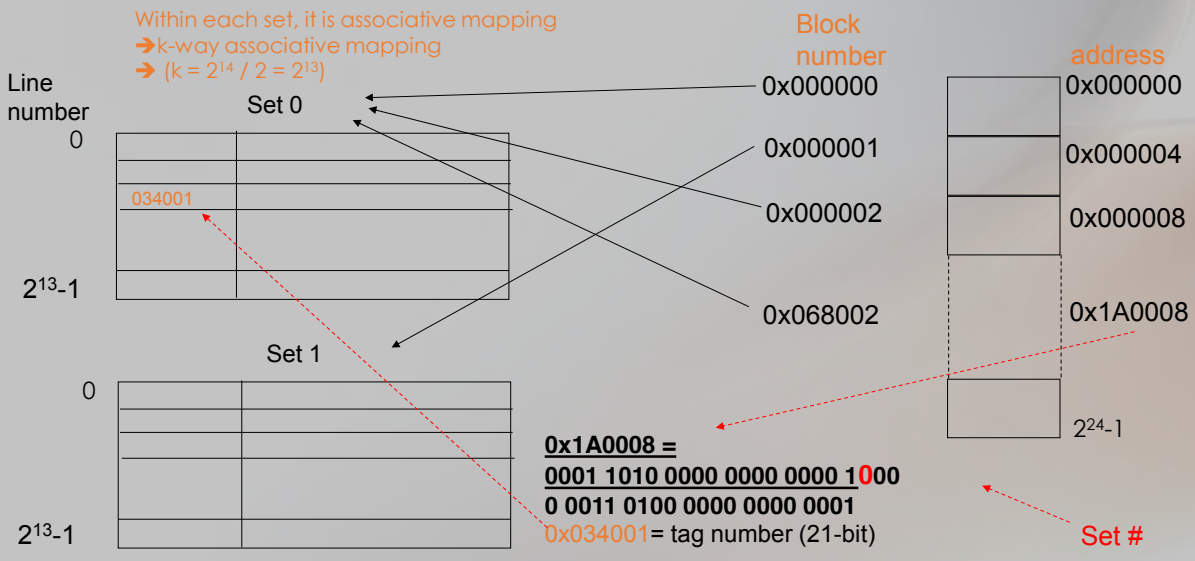
45

Set Associative Mapping Concept



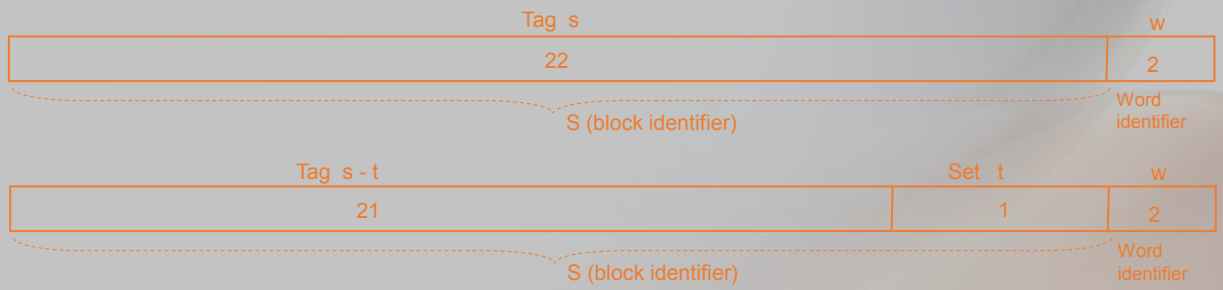
46

Set Associative Mapping Illustrated



47

Set Associative Mapping Address Structure - I



- Some tag bits in fully associative mapping is now used for set number
  - E.g., If you have 2 sets, 1 bit is used for determining set #.
    The rest (21 bits) is used for Tag bits
  - E.g., If you have 1024 sets, 10 bits are used for determining set #
    The rest (22-10 = 12 bits) is used for tag bits

48

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

49

Replacement Algorithms - Associative & Set Associative

- **Least Recently Used (LRU)**
  - replace block that has been in cache **longest with no reference** to it
    - e.g. in 2-way set associative: USE bit
  - Best hit ratio
- **First In First Out (FIFO)**
  - replace block that has been **in cache longest**
  - Easy to implement with RR or circular buffer
- **Least Frequently Used (LFU)**
  - replace block which has had **fewest hits**
  - Use a counter for each line
- **Random**
  - not intuitive? It is shown not bad
- Must be implemented in Hardware for speed

50

Write policy - Handling Writes

- What if a cache line has been **modified**?
  - There will be **inconsistency** between cache and memory
  - If you pick this cache line for replacement, you cannot just overwrite it unless its main memory is updated
- How do we avoid inconsistency?
  - Always write data on both cache and memory: **Write-through**
  - Copy the modified cache to memory only when the line is replaced: **Write-back**

51

Write Policy 1 - Write through

- Simplest solution
- All writes go to main memory as well as cache
  - Main memory is also up to date
- Disadvantages
  - Lots of traffic. All write traffic must go to memory.
  - **Slow**. Cache is basically **useless** for writing.
- How slow?
  - In SPEC2000, **10%** are store instructions. If it requires 100 clock cycles for memory write, the CPI (Cycles Per Instruction) for write = 1 * 90% + (1 + 100) * 10% = **11**

52

Write Policy 2 - Write back

- **Updates** initially made in **cache only** (no memory write)
  - **Update** bit for cache line is set when update occurs
  - If block is to be replaced, write to main memory only if update bit is set
- Disadvantages
  - Discrepancy exists between cache and memory for some duration
  - Requires complex circuitry, and potential bottleneck

53

Issues with Policy 2 (Write back)

- Competition with I/O (DMA)
  - Option1: I/O (DMA) must access the main memory through cache (no one writes directly on memory. Cache change is first)
  - Option 2: I/O (DMA) may address main memory directly. If the matching cache content is modified, the memory is invalid, and I/O module should not write on memory, and vice versa
- Multiple processors may have individual caches
  - Each processor changes only its own cache. Whose cache is correct?
  - **Cache coherence problem**

54

Which is better?

- Write through or Write back?
  - write-back seems better, but it depends on the frequency of write operation
  - If write is infrequent, write-through may be faster!
- N.B. **15%** of memory references are **writes**. For high-performance computing (HPC), 33% or even 50%.

55

Comparison example

- Assumptions
  - Each word = 4 bytes
    - 30 *ns* for 4-byte word transfer
  - Cache line (block) size = 32 byte = 8 words
    - i.e., total transfer time for each block = 8 words * 30 *ns* = 240 ns
- Which is better between write-through and write-back?
  - For each write operation, they spend
    - Write-back (must write **entire block** of 8 words): 240 ns
    - Write-through (can write **individual word**): 30 ns
  - If a cache line gets written more than 8 times before swap-out, write-back is better. (always 240 ns)
  - Easier answer: **Each line is 8 words**, and the unit of transfer is word. So the decision threshold is 8 words.

56