Chapter 2: Memory Unit

2.1 Introduction

Although seemingly simple in concept, computer memory exhibits perhaps the widest range of type, technology, organization, performance, and cost of any feature of a computer system. No one technology is optimal in satisfying the memory requirements for a computer system. As a consequence, the typical computer system is equipped with a hierarchy of memory subsystems, some internal to the system and some external.

This chapter focuses on internal memory elements. The first section examines key characteristics of computer memories. The remainder of the chapter examines: cache memory and main (RAM/ Internal) memory.

2.2 Computer Memory System Overview

2.2.1 Characteristics of Memory Systems

The subject of computer memory is made more manageable if we classify memory systems according to their key characteristics, as shown below.

Location

The term location refers to whether memory is internal or external to the computer.

Internal Memory:

- It is directly accessible by the processor.
- Often equated with main memory, but there are other forms of internal memory:
    - Cache is another form of internal memory
    - The processor has its own local memory, in the form of registers
    - The control unit portion of the processor may have its own internal memory – control memory

External Memory:

- Also called secondary memory
- Consists of peripheral storage devices, such as disk and tape, that are accessible to the processor via I/O controllers (module).

Capacity

Refers to the amount of information that can be contained in a memory unit. For internal memory, this is typically expressed in terms of bytes or words. Common word lengths are 8, 16 and 32 bits. External memory capacity is typically expressed in terms of bytes.

Word: Natural unit of organization of memory. The size of the word is typically equal to the number of bits used to represent a number and to the instruction length.

Addressable unit: Smallest location which can be uniquely addressed – in some systems it is a word, however many systems allow addressing at the byte level.

Unit of Transfer

For internal memory, the unit of transfer is equal to the number of data lines into and out of the memory module. This may be equal to the word length, but is often larger, such as 64,128, or 256 bits.

For external memory, data are often transferred in much larger units than a word, and these are referred to as blocks.

Access Methods
Refers to the method of accessing units of data. These include the following:

Sequential Access
- Start at the beginning and read through in order
- Access time depends on location of data and previous location
- Example, tape drive units

Direct Access
- Individual blocks have unique address
- Access is done using a combination of moving to a general memory “area” followed by a sequential searching to reach the final location
- Access time depends on location of data and previous location
- Example, disk drive units

Random Access
- Individual addresses identify location exactly
- Access time is independent of data location and previous location
- Example, main memory/RAM

Associative Access
- A variation of random access memory
- Data items are accessed based on their contents rather than their address
- Search all data items in parallel for a match
- Access time is independent of data location and previous location
- Example, some cache memory units

Performance
Access time (latency)
- The time between presenting an address and getting access to valid data

For random-access memory, this is the time it takes to perform a read or write operation, that is, the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use. For non-random-access memory, access time is the time it takes to position the read-write mechanism at the desired location.

Memory Cycle time
- primarily applied to random-access memory
- Consists of access time plus recovery time
- Time may be required for the memory to “recover” before the next access

Transfer rate
- The rate at which data can be transferred into or out of a memory unit

For non-random access memory, the following relationship holds:

$$T_N = T_A + N/R$$

Where

T_N= Average time to read or write N bits
T_A=Average access time
N= Number of bits
R= Transfer rate, in bits per seconds (bps)

Physical Types

A variety of physical types of memory have been employed. The most common today are:
Semiconductor – RAM
Magnetic – disk and tape
Optical – CD and DVD
Magneto-optical

Physical Characteristics

Several physical characteristics of data storage are important. In a volatile memory, information is lost when electrical power is switched off. In a nonvolatile memory, information once recorded remains without deterioration until deliberately changed; no electrical power is needed to retain information. Magnetic-surface memories are nonvolatile. Semiconductor memory may be either volatile or nonvolatile. Non erasable memory cannot be altered, except by destroying the storage unit. Semiconductor memory of this type is known as read only memory (ROM).

Organization

- The physical arrangement of bits to form words
- The obvious arrangement is not always used

2.2.2 The Memory Hierarchy

The design constraints on a computer's memory can be summed up by three questions:
How much? How fast? How expensive?

How much?
- If the capacity is there, applications will be developed to use it.

How fast?
- To achieve performance, the memory must be able to keep up with the processor.

How expensive?
- For a practical system, the cost of memory must be reasonable in relationship to other components

There is a trade-off among the three key characteristics of memory: capacity, access time, and cost.
Faster access time – greater cost per bit
Greater capacity – smaller cost per bit
Greater capacity – slower access time

The way out of this dilemma is not to rely on a single memory component or technology. Employ a memory hierarchy.

As one goes down the hierarchy:
a) decreasing cost per bit
b) increasing capacity
c) increasing access time
d) decreasing frequency of access of the memory by the processor.

Thus smaller, more expensive, faster memories are supplemented by larger, cheaper, slower memories. The key to the success of this organization is item (d).

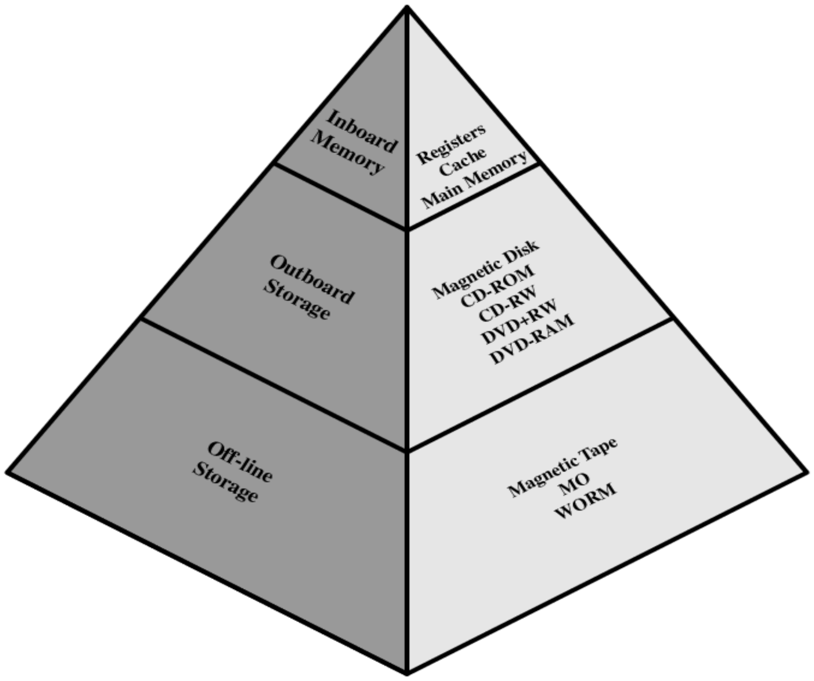

Figure 1: The memory hierarchy

The basis for the validity of condition (d) is a principle known as **locality of reference**:

- This principle says that memory references by the processor, for both data and instructions, cluster
    - Programs contain iterative loops and subroutines - once a loop or subroutine is entered, there are repeated references to a small set of instructions
    - Operations on tables and arrays involve access to a clustered set of data word
- Keep these clusters in high speed memory to reduce the average delay in accessing data

2.3 Cache Memory

2.3.1 Cache Memory Principles

Cache memory

- Compared to the size of main memory, it is relatively small
- Operates at or near the speed of the processor
- Placed between the processor and main memory
- Located either on the processor chip or on a separate module
- Contains copies of sections of main memory
- Because of the high speeds involved with the cache, management of the data transfer and storage in the cache is done in hardware – the O/S does not "know" about the cache

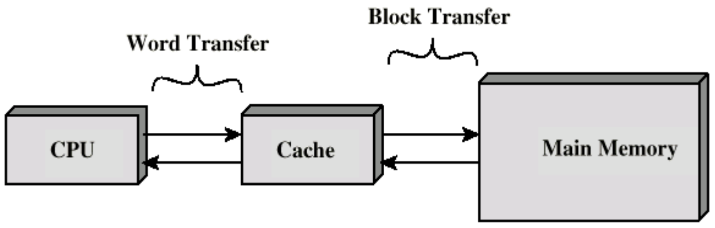

Figure 2: Cache and main memory

Cache Operation Overview

- Processor requests the contents of some memory location
- The cache is checked for the requested data
    - If found, the requested word is delivered to the processor
    - If not found, a block of main memory containing the requested word is read into the cache and the requested word is delivered to the processor

When a block of data is fetched into the cache to satisfy a single memory reference, it is likely that there will be future references to that same memory location or to other words in the block – locality of reference principle.

Figure 3 below depicts the structure of a cache-main memory system. Main memory consists of up to 2^n addressable words, with each word having a unique n bit address. For mapping purposes, this memory is considered to consist of a number of fixed-length blocks of K words each. That is, there are $M = 2^n/K$ blocks.

Cache consists of C lines of K words each, and the number of lines is considerably less than the number of main memory blocks (C << M). At any time, some subset of the blocks of memory resides in lines in the cache. If a word in a block of memory is read, that block is transferred to one of the lines of the cache. Because there are more blocks than lines, an individual line cannot be uniquely and permanently dedicated to a particular block. Thus, each line includes a tag that identifies which particular block is currently being stored. The tag is usually a portion of the main memory address, as described later in this section.

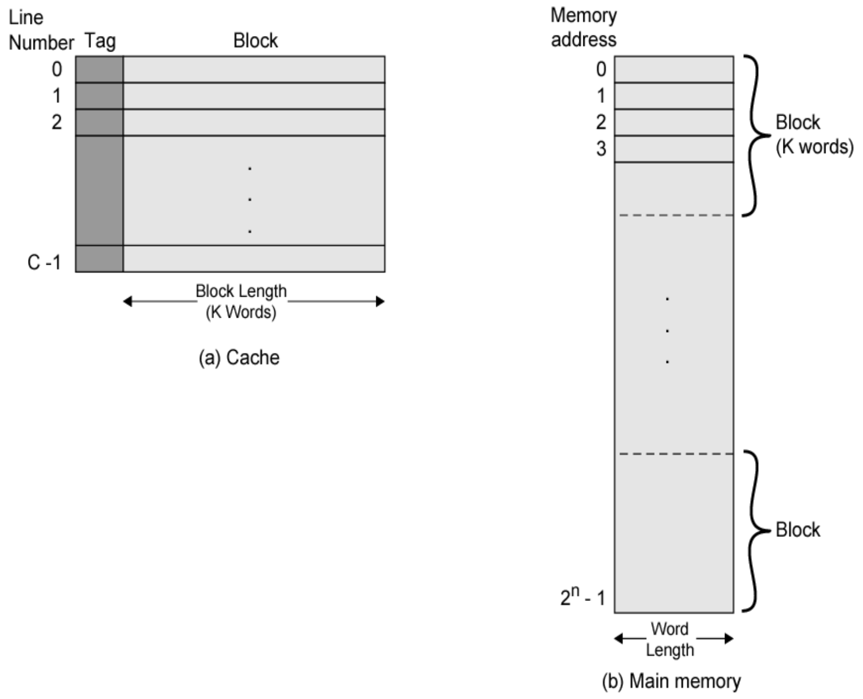

Figure 3: Cache/Main Memory Structure

An example of a typical cache organization is shown below:

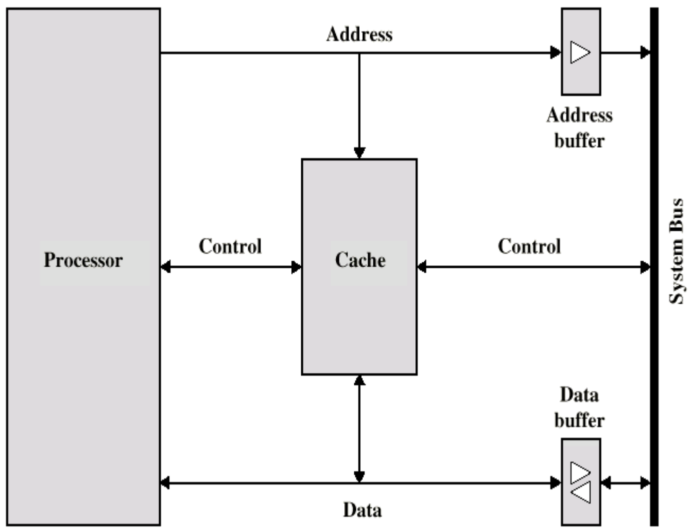

Figure 4: Typical cache organization

2.3.2 Elements of Cache Design

Table 1 below lists basic design elements that serve to classify and differentiate cache architectures.

Cache Size	**Write Policy**
Mapping Function	Write through
Direct	Write back
Associative	Write once
Set associative	**Line Sizes**
Replacement Algorithms	**Number of Caches**
Least recently used (LRU)	Single or two levels
First in first out (FIFO)	Unified or split
Least frequently used (LFU)	
Random	

Table 1: Elements of cache design

Cache Size

- ➢ Small enough so overall cost/bit is close to that of main memory
- ➢ Large enough so overall average access time is close to that of the cache alone
- ➢ Large caches tend to be slightly slower than small caches
- ➢ Also limited by the available chip and board area

Mapping Function

An algorithm is needed to map main memory blocks into cache lines. A method is needed to determine which main memory block occupies a cache line. The choice of the mapping function dictates how the cache is organized.

Three techniques used: Direct, Associative, and Set associative

Assume the following:

- Cache of 64 Kbytes, organized as 16K (2^{14}) lines of 4 bytes each
- Transfers between main memory and cache are in blocks of 4 bytes each
- Main memory of 16 Mbytes, with each byte directly addressable by a 24-bit address (where 2^{24} = 16M) – for mapping purpose, assume main memory consists of 4M blocks of 4 bytes each

Direct Mapping

- ➢ Each block of main memory maps to only one cache line:
  - I = j modulo m
  - Where, I = cache line number; j = main memory block number and m= number of lines in the cache

Cache Line	Main memory blocks assigned
0	0, m, 2m, …
1	1, m+1, 2m +1, …
….	
m-1	m-1,2m-1,3m-1, …

Table 2: Direct mapping cache line table

- ➢ The mapping function is easily implemented using main memory address. The main memory is viewed as having three fields: Word identifier, Line identifier, Tag identifier
  - Least significant w bits identify a unique word or byte within a block
  - most significant s bits are split into two:
    - a cache line field of r bits --- identifies one of the m = 2^r lines of the cache, that should hold the data
    - a tag field of s-r bits (most significant) --- stored in the cache along with the data words of the line and identifies a particular block of main memory

- For every memory reference that the CPU makes, the specific line that would hold the reference (if it has already been copied into the cache) is determined
    - The tag held in that line is checked to see if the correct block is in the cache
- Example of Direct Mapping for a 24 bit address

Tag (s-r)	Line (r)	Word (w)
8 bits	14 bits	2 bits

    - 2 bit word identifier , 22 bit block identifier , 8 bit tag (22-14) , 14 bit line
    - 8 bit tag stored with the 32 bit block of data
- Observe that no two blocks that can be mapped into a given line could have the same Tag field

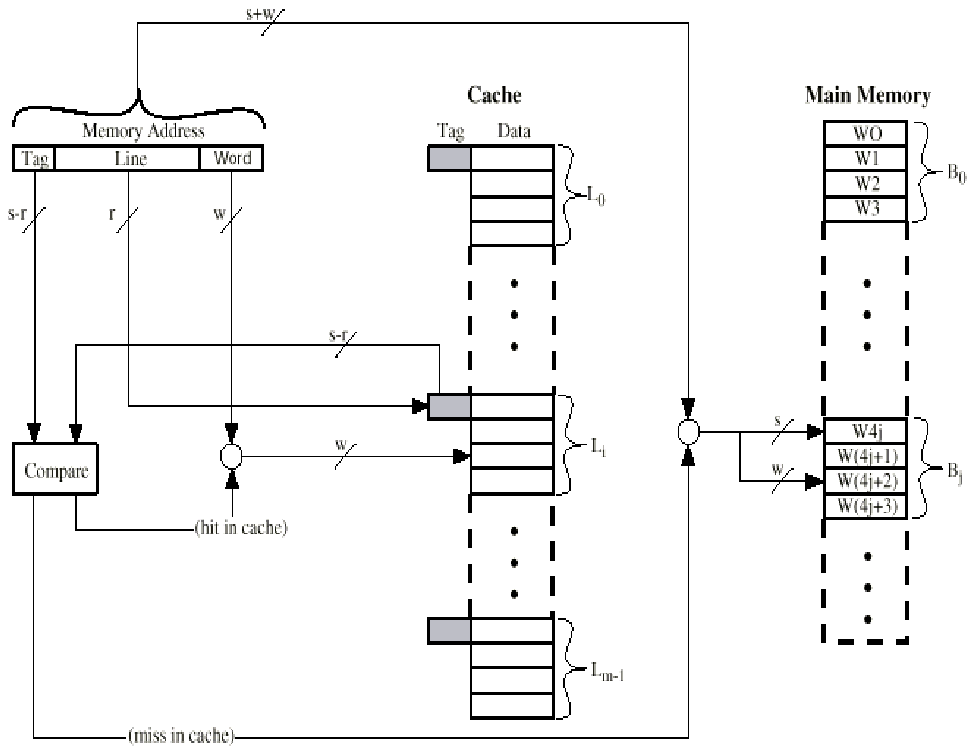

Figure 5: Direct mapping cache organization

Direct Mapping Summary

- Address length = (s+w) bits
- Number of addressable units = $2^{(s+w)}$ words or bytes
- Block size = line size = 2^w words or bytes
- Number of blocks in main memory = $2^{(s+w)}/2^w = 2^s$
- Number of lines in cache = m = 2^r
- Size of tag = (s-r) bits

Advantages of direct mapping

- Easy to implement
- Relatively inexpensive to implement
- Easy to determine where a main memory reference can be found in cache

Disadvantages of direct mapping

- Each main memory block is mapped to a specific cache line
- Through locality of reference, it is possible to repeatedly reference to blocks that map to the same line number
    - These blocks will be constantly swapped in and out of cache, causing the hit ratio to be low

Associative Mapping

- A main memory block can be loaded into any line of the cache
- A memory address is interpreted as a tag and a word field
- The tag field uniquely identifies a block of main memory
- Each cache line's tag is examined simultaneously to determine if a block is in cache

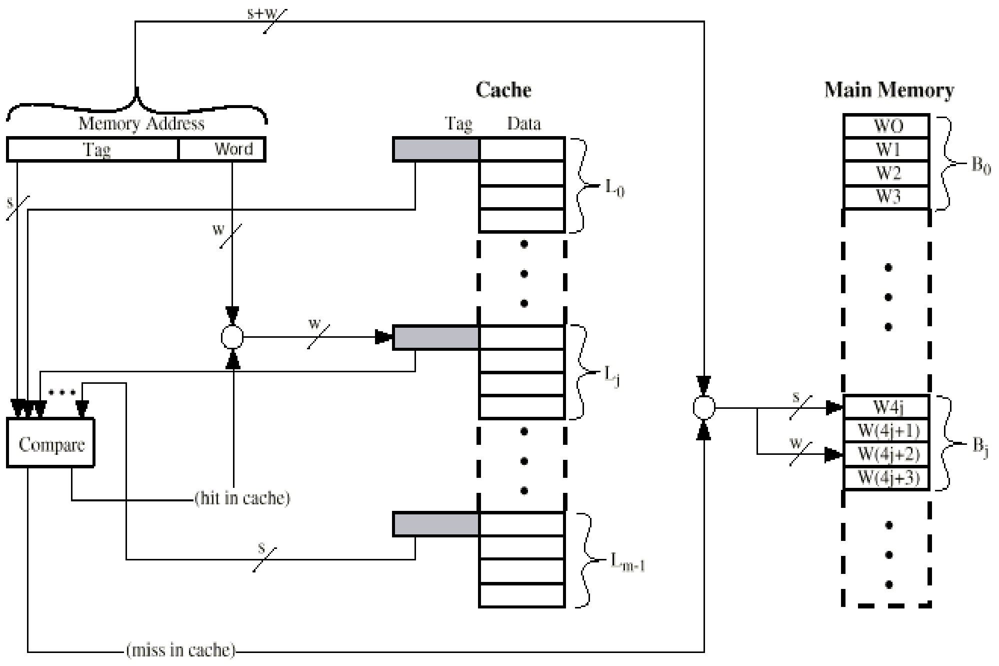

Figure 6: Associative mapping cache organization

- Example of associative mapping for a 24 bit address
    - Main memory addresses are viewed as two fields

Tag (s)	Words (w)
22 bits	2 bits

    - 22 bit tag stored with the 32 bit block of data
    - Tag field compared with tag entry to check for cache hit

Associative Mapping Summary

- Address length = (s+w) bits
- Number of addressable units = $2^{(s+w)}$ words or bytes
- Block Size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{(s+w)}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

Associative Mapping Pros and Cons

- Flexibility as to which block to replace when a new block is read into cache
    - Replacement algorithms designed to maximize cache hit ratio
- Complex circuitry required to examine the tags of all cache lines in parallel

Set Associative Mapping

- Compromise between direct and associative mapping, builds on the strengths of both
- Cache divided into v sets, each set contains k lines. The relationships are:
    - m =v * k
    - i= j modulo v
    - where i = cache set number, j=main memory block number and m= number of lines in the cache

- A given block maps into any line in a given set
  - Block B_i can be mapped into any line in set i
  - Assume that k = 2 – meaning there are 2 lines per set
    - 2-way associative mapping – k-way mapping
    - A given block can map into either of the 2 lines in exactly 1 set
- Main memory addresses are viewed as three fields:
  - A tag field, a set field and a word field

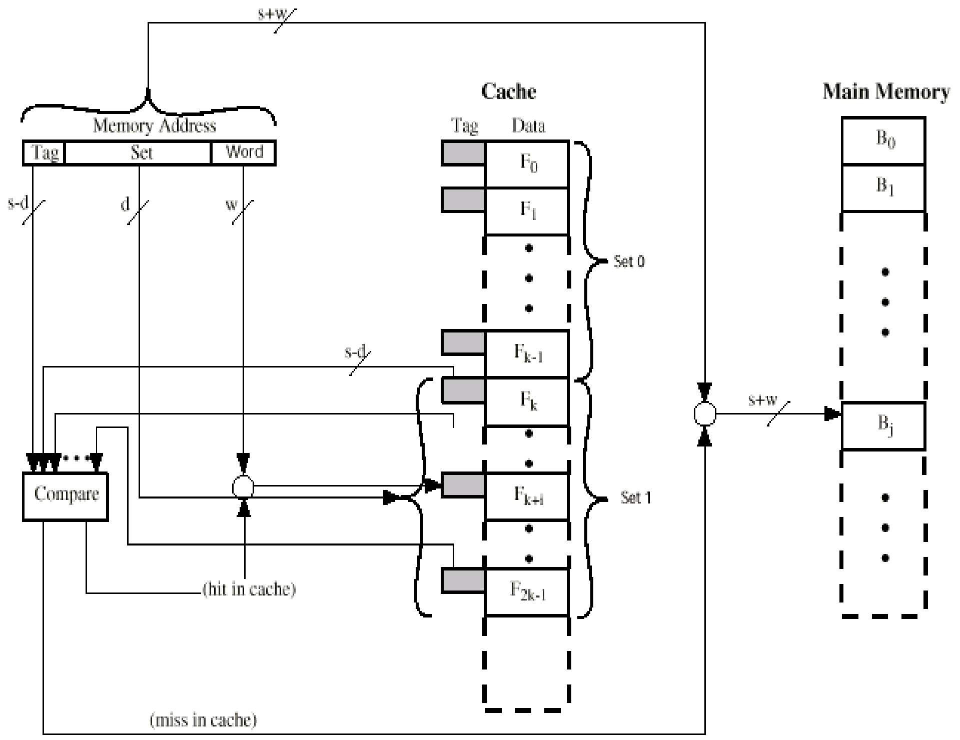

Figure 6: k-Way set associative cache organization

- Example of set associative mapping for 24 bit address

Tag (s-d)	Set (d)	Word (w)
9 bits	13 bits	2 bits

  - 9 bit tag field – only compared with k tags in the given set
  - 13 bit set field – specifies one of the v = 2^d sets

Set Associative Mapping Summary

- Address length = (s+w) bits
- Number of addressable units = $2^{(s+w)}$ words or bytes
- Block Size = line size = 2^w words or bytes
- Number of blocks in main memory = $2^{(s+w)}/2^w = 2^s$
- Number of lines in set = k
- Number of sets v = 2^d
- Number of lines in cache = kv = $k*2^d$
- Size of tag = (s-d) bits
- v = m, k = 1 reduces to direct mapping
- v = 1, k = m reduces to associative mapping
- 2-way mapping is most commonly used – significantly improves cache hit ratio over direct mapping
- 4-way mapping – makes a modest additional improvement for a relatively small additional cost

Replacement Algorithms

When a new block is brought into the cache, one of the existing blocks must be replaced using some replacement algorithm.

For Direct Mapping
- No choice
- Each block only maps to one line
- Must replace that line

For Associative and Set Associative Mappings
- The algorithms must be implemented in hardware for speed
- Least Recently Used (LRU)
  - Most effective
  - Replace the block in the set that has been in cache the longest with no references to it
  - For a 2-way set associative – each line includes a USE bit
- First-in-first-out (FIFO)
  - Replace the block in the set that has been in the cache the longest
  - Uses a round-robin or circular buffer technique
- Least Frequently Used (LFU)
  - Replace the block in the set that has experienced the fewest references
  - Associate a counter with each line
- Pick a line at random – not based on usage
  - Only slightly inferior in performance to algorithms based on usage

Write Policy

When a line is to be replaced, we must update the original copy of the line in main memory if any addressable unit in the line has been changed. A variety of write policies with performance and economic trade-offs, exist.

But, there are two problems to contend with.
- What if multiple processors are present and each one has its own cache
- What if an I/O device addresses main memory directly

Write Through
- Anytime a word in cache is changed, it is also changed in main memory
- Both copies always agree
- Generates a lot of memory traffic
- Each processor can monitor main memory activity to keep local cache current

Write Back
- Minimizes memory writes
- Updates are made only in cache
- UPDATE bit associated with the line is set
- When the block is replaced it is written back to main memory only if the UPDATE bit is set
- This may cause other caches to be out of sync
- I/O must access main memory through the cache

Line Sizes
- How much data should be transferred from main memory to the cache in a single memory reference
- Complex relationship between block size and hit ratio as well as the operation of the system bus itself
- As block size increases,

- Locality of reference predicts that the additional information transferred will likely be used and thus increases the hit ratio (good)
- Number of blocks in cache goes down, limiting the total number of blocks in the cache (bad)
- As the block size gets big, the probability of referencing all the data in it goes down (hit ratio goes down) (bad)
- Size of 4-8 addressable units seems about right for current systems

<u>Number of caches</u>

- Single vs. 2-level
    - Modern CPU chips have on-board cache (L1)
        - 80486 -- 8KB
        - Pentium -- 16 KB
        - Power PC -- up to 64 KB
    - L1 provides best performance gains
    - Secondary, off-chip cache (L2) provides higher speed access to main memory
    - L2 is generally 512KB or less -- more than this is not cost-effective
- Unified vs. split cache
    - Unified cache stores data and instructions in 1 cache
        - Only 1 cache to design and operate
        - Cache is flexible and can balance “allocation” of space to instructions or data to best fit the execution of the program -- higher hit ratio
    - Split cache uses 2 caches -- 1 for instructions and 1 for data
        - Must build and manage 2 caches
        - Static allocation of cache sizes
        - Can outperform unified cache in systems that support parallel execution and pipelining

Assignment I

Chapter 4 (Cache Memory)
Review Questions
4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.11, 4.12