Memory Systems

Computer Memory System Overview

- Historically, the limiting factor in a computer's performance has been memory access time
  - Memory speed has been slow compared to the speed of the processor
  - A process could be bottlenecked by the memory system's inability to "keep up" with the processor

Computer Memory System Overview

Terminology

- **Capacity**: (For internal memory) Total number of words or bytes. (For external memory) Total number of bytes.
- **Word**: the natural unit of organization in the memory, typically the number of bits used to represent a number - typically 8, 16, 32
- **Addressable unit:** the fundamental data element size that can be addressed in the memory -- typically either the word size or individual bytes
- **Access time:** the time to address the unit and perform the transfer
- **Memory cycle time:** Access time plus any other time required before a second access can be started

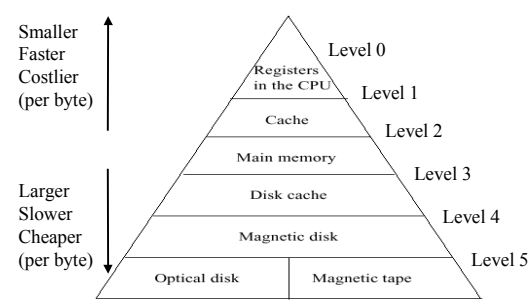
Memory Hierarchy

- **Major design objective of any memory system**
  - To provide adequate storage capacity at
  - An acceptable level of performance
  - At a reasonable cost
- **Four interrelated ways to meet this goal**
  - Use a hierarchy of storage devices
  - Develop automatic space allocation methods for efficient use of the memory
  - Through the use of virtual memory techniques, free the user from memory management tasks
  - Design the memory and its related interconnection structure so that the processor can operate at or near its maximum rate

Basis of the memory hierarchy

- Registers internal to the CPU for temporary data storage (small in number but very fast)
- External storage for data and programs (relatively large and fast)
- External permanent storage (much larger and much slower)
- Remote Secondary Storage (Distributed File Systems, Web Servers)

The Memory Hierarchy



Typical Memory Parameters

Memory Type	Technology	Size	Access Time
Cache	Semiconductor RAM	128-512 KB	10 ns
Main Memory	Semiconductor RAM	4-128 MB	50 ns
Magnetic Disk	Hard Disk	Gigabyte	10 ms, 10 MB/sec
Optical Disk	CD-ROM	Gigabyte	300 ms, 600 KB/sec
Magnetic Tape	Tape	100s MB	Sec-min., 10MB/min

Typical Memory Parameters

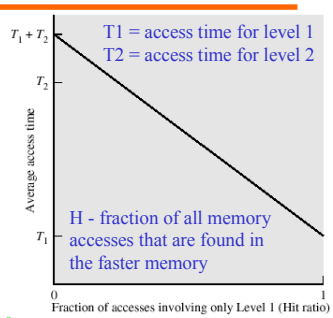
Suppose that the processor has access to two levels of memory. Level 1 contains 1000 words and has an access time of 0.01 µs; level 2 contains 100,000 words and has an access time of 0.1 µs. Assuming that

if a word to be accessed is in level 1, then the processor access it directly.

If it is in level 2, then the word is first transferred to level 1 and then accessed by the processor.

For simplicity, ignore the time required for the processor to determined whether is in level 1 or level 2. A typical performance of a simple two level memory has this shape:

Typical Memory Parameters



Suppose 95% of the memory accesses are found in level 1 the average time to access a word is

(0.95)(0.01 µs) +

(0.005)(0.01µs + 0.1µs) =

0.015 µs

The Locality Principle

The memory hierarchy works because of *locality of reference*

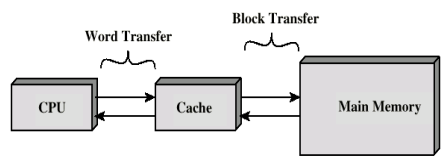
- Well written computer programs tend to exhibit good locality. That is, they tend to reference data items that are near other recently referenced data items, or that were recently referenced themselves. This tendency is known as the locality principle.
- All levels of modern computer systems, from the hardware, to the operating system, to the application programs, are designed to exploit locality.

The Locality Principle

- At hardware level, the principle of locality allows computer designers to speed up main memory accesses by introducing small fast memories known as the *cache memories*.
- At operating system level, main memory is used to cache the most recently referenced chunks of virtual address space and the most recently used disk blocks in a disk file system.
- At application level, Web browsers cache recently referenced documents in local disk

Cache Memory

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module
- Intended to achieve high speed at low cost

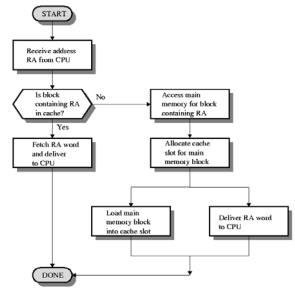


Cache Memory

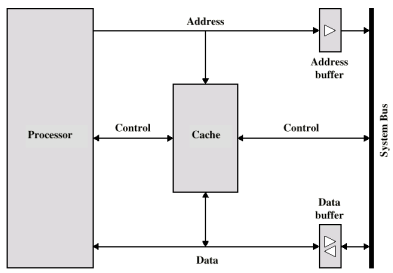
- Cache retains copies of recently used information from main memory, it operates transparently from the programmer, automatically decides which values to keep and which to overwrite.
- An access to an item which is in the cache: **hit**
- An access to an item which is not in the cache: **miss**
- The proportion of all memory accesses that are found in cache: **hit rate**

Cache operation - overview

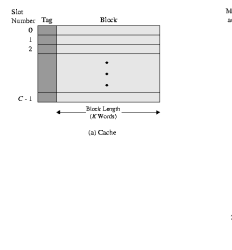
- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU



Typical Cache Organization



Cache/Main Memory Structure



- Main memory consists of fixed length blocks of **K** words (**M = 2^n/K** blocks)
- Cache consists of **C *Lines*** of **K** words each
- The number of lines is much less than the number of blocks (**C << M**)
- Block size = Line Size

Cache includes **tags** to identify which block of main memory is in each cache slot

Mapping Function

- Fewer cache line than main memory block
- Need to determine which memory block currently occupies a cache line
- Need an algorithm to map memory block to cache line
- Three Mapping Techniques:
  - **Direct**
  - **Associative**
  - **Set associative**

Direct Mapping

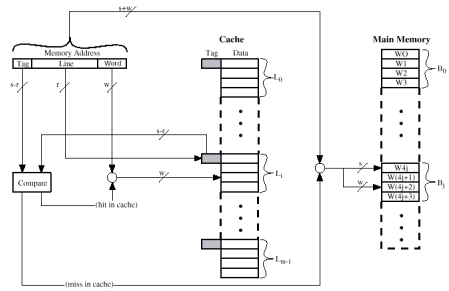
- Each main memory address can be viewed as consisting 3 fields:
  - The least significant *w* bits identify a unique word or byte within a block of main memory
  - The remaining *s* bits specify one of 2^s blocks of main memory
  - The cache logic interprets these s bits as:
    - a tag field of *s - r* bits (most significant portion)
    - a line field of *r* bits

s - r	*r*	*w*

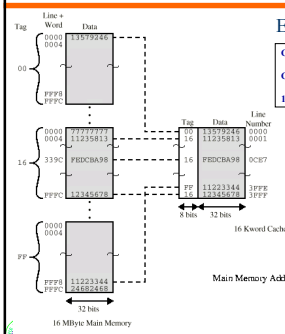
Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2^s-m
1	1,m+1, 2m+1… 2^s-m+1
…	…
m-1	m-1, 2m-1,3m-1… 2^s -1

m=2^r line of cache

Direct Mapping



Direct Mapping



Example System:

Cache of 64kByte

Cache block of 4 bytes - i.e. cache is 16k (2^{14}) lines of 4 bytes

16 MBytes main memory - 24 bit address (2^{24}=16M)

Cache line	Starting memory address of block
0	000000, 010000, …, FF0000
1	000004, 010004, …, FF00004
…	…
m-1	00FFFC, 01FFFC, …, FFFFC

Main Memory Address =

Tag	Line	Word
8	14	2

Direct Mapping

Memory Address	Tag	Cache Line	Word
FFF9CA			
81FCAE			

Main Memory Address =

Tag	Line	Word
8	14	2

Direct Mapping

Example:

Memory size 1MB (20 address bits) addressable to individual bytes

Cache size of 1K lines, each 8 bytes

Word id = 3 bits

Line id = 10 bits

Tag id = 7 bits

Where is the byte stored at main memory location ABCDE stored in the cache

Cache Line #

Word location

Tag id

Direct Mapping

Simple

Inexpensive

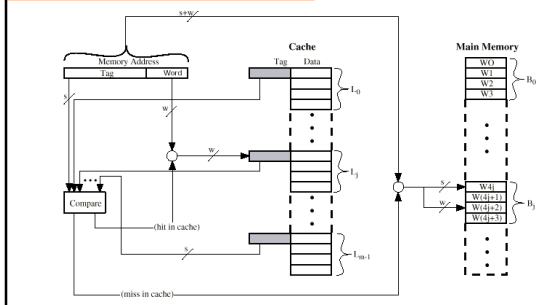
Fixed location for given block

If a program accesses 2 blocks that
map to the same line repeatedly, cache misses are very high

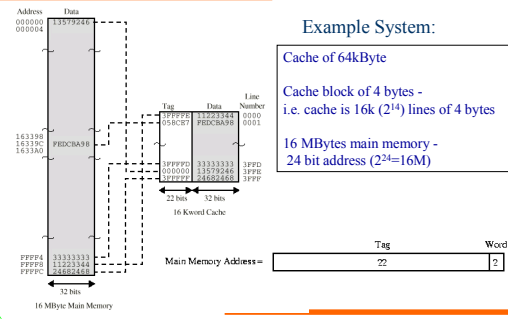
Associative Mapping

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive

Associative Mapping

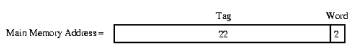


Associative Mapping



Associative Mapping

Memory Address	Cache Tag	Cache Word
FFF9CA		
81FCAE		



Associative Mapping

Example:

Memory size 1MB (20 address bits) addressable to individual bytes

Cache size of 1K lines, each 8 bytes

Word id = 3 bits

Tag id = 17 bits

Where is the byte stored at main memory location ABCDE stored in the cache

Word location

Tag id

Set Associative Mapping

- Cache is divided into a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set

Address length = (s + w) bits

Number of addressable units = 2^{s+w} bytes or words

Block size = line size = 2^w bytes or words

Number of blocks in main memory = $(2^{s+w})/2^w = 2^s$

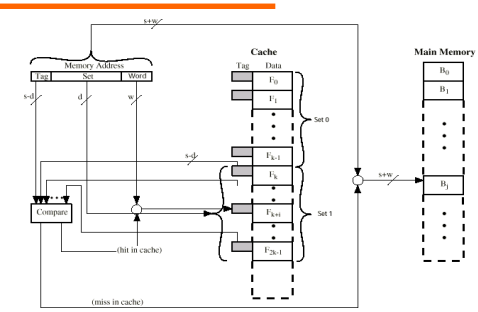
Number of lines in set = k

Number of sets v = 2^d

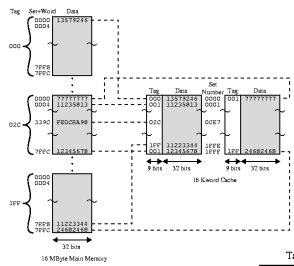
Number of lines in cache = kv = k x 2^d

Size of tag = (s - d) bits

Set Associative Mapping



Set Associative Mapping



Cache of 64kByte

Cache block of 4 bytes - i.e. cache is 16k (2^{14}) lines of 4 bytes

16 MBytes main memory - 24 bit address (2^{24}=16M)

2 lines in each set
16k/2 = 8k set

	Tag	Set	Word
Main Memory Address =	9	13	2

Set Associative Mapping

Use set field to determine cache set to look in
Compare tag field to see if we have a hit, e.g

Memory		Cache	
Address	**Tag**	**Set number**	**word**
FFF9CA			
81FCAE			

	Tag	Set	Word
Main Memory Address =	9	13	2

Set Associative Mapping

Example:
Memory size 1MB (20 address bits) addressable to individual bytes
Cache size of 1K lines, each 8 bytes

4-way set associative mapping 1024/4 = 256 sets

Word id = 3 bits Set id = 8 bit Tag id = 17 bits

Where is the byte stored at main memory location ABCDE stored in the cache

Word location
Set
Tag

Replacement Algorithms

- When a new block is brought into the cache, one of the existing blocks must be replaced .
- Direct Mapping: One possible line for any particular block - No choice
- Associative/Set Associative Mapping:
  - Least Recently used (LRU): Replace block that has not been referenced the longest. E.g. in 2 way set associative, Which of the 2 block is LRU?
  - First in first out (FIFO): replace block that has been in cache longest
  - Least frequently used: replace block which has had fewest hits
  - Random

Write Policy

- Before a block that is resident in the cache can be replaced, it is necessary to consider whether it has been altered in the cache but not in the main memory.
- If it has not (been altered in cache), then the old block in the cache can be overwritten.
- If it has (been altered in cache), it means at least one write operation has been performed on a word in that cache line and main memory must be updated accordingly.

Write Policy

Write Through:
All writes go to main memory as well as cache
Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
Lots of traffic
Slows down writes

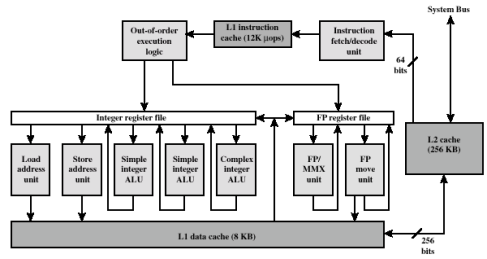
Write Back
Updates initially made in cache only
Update bit for cache slot is set when update occurs
If block is to be replaced, write to main memory only if update bit is set
Other caches get out of sync
I/O must access main memory through cache
N.B. 15% of memory references are writes

Line Size

Larger blocks reduce the number of blocks that fit into the cache.

As block becomes larger, each additional word is farther from the requested word, therefore less likely to be needed in the near future

Pentium 4 Cache



PowerPC Cache