Memory and Caching

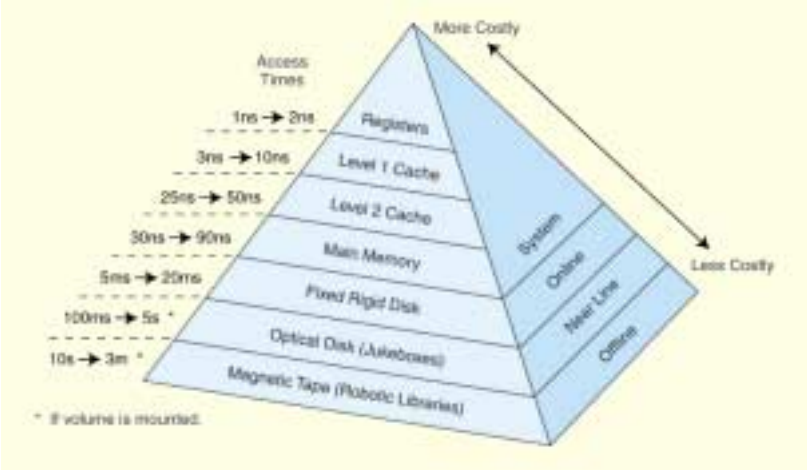
Chapter 6

Chapter 6 Objectives

- Master the concepts of hierarchical memory organization.
- Understand how each level of memory contributes to system performance, and how the performance is measured.
- Master the concepts behind cache memory
  - Skipping virtual memory, memory segmentation, paging and address translation. Cover in OS class.

The Memory Hierarchy

- This storage organization can be thought of as a pyramid:



Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

- As one goes down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access of the memory by the processor – locality of reference

Locality of Reference

- Temporal Locality
  - Programs tend to reference the same memory locations at a future point in time
  - Due to loops and iteration, programs spending a lot of time in one section of code
- Spatial Locality
  - Programs tend to reference memory locations that are near other recently-referenced memory locations
  - Due to the way contiguous memory is referenced, e.g. an array or the instructions that make up a program
  - Sequential Locality
    - Instructions tend to be accessed sequentially
- Locality of reference does not always hold, but it usually holds

Semiconductor Memory

- RAM – Random Access Memory
  - Misnamed as all semiconductor memory is random access
  - Read/Write
  - Volatile
  - Temporary storage
  - Two main types: **Static** or **Dynamic**

Dynamic RAM

- Bits stored as charge in semiconductor capacitors
- Charges leak
- Need refreshing even when powered
- Simpler construction
- Smaller per bit
- Less expensive
- Need refresh circuits (every few milliseconds)
- Slower
- Main memory

Static RAM

- Bits stored as on/off switches via flip-flops
- No charges to leak
- No refreshing needed when powered
- More complex construction
- Larger per bit
- More expensive
- Does not need refresh circuits
- Faster
- Cache

So you want fast?

- It is possible to build a computer which uses only static RAM (the memory used to build a cache)
- This would be very fast
- This would need no cache
  - How can you cache cache?
- This would cost a very large amount

Read Only Memory (ROM)

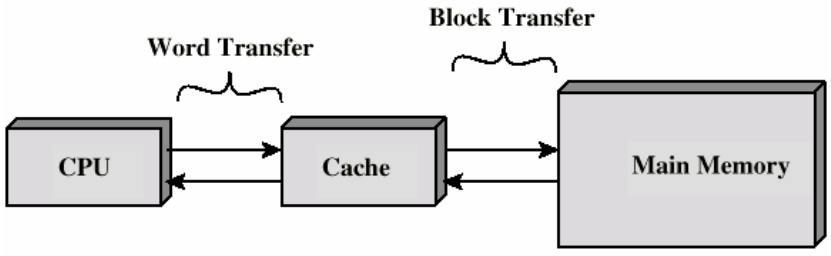
- Permanent storage
- Microprogramming
- Library subroutines
- Systems programs (BIOS)
- Function tables

Types of ROM

- Written during manufacture
  - Very expensive for small runs
- Programmable (once)
  - PROM
  - Needs special equipment to program
- Read “mostly”
  - Erasable Programmable (EPROM)
    - Erased by UV
  - Electrically Erasable (EEPROM)
    - Takes much longer to write than read
  - Flash memory
    - Erase whole memory electrically

Cache

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module
  - An entire blocks of data is copied from memory to the cache because the principle of locality tells us that once a byte is accessed, it is likely that a nearby data element will be needed soon.



Cache operation - overview

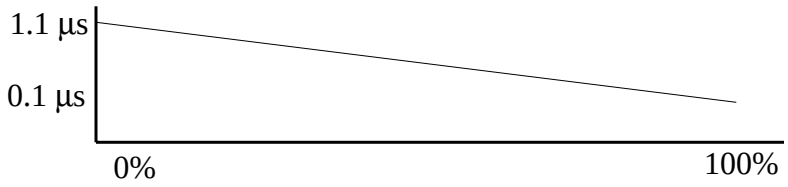
- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Cache Definitions

- This leads us to some definitions.
  - A *hit* is when data is found at a given memory level.
  - A *miss* is when it is not found.
  - The *hit rate* is the percentage of time data is found at a given memory level.
  - The *miss rate* is the percentage of time it is not.
  - Miss rate = 1 - hit rate.
  - The *hit time* is the time required to access data at a given memory level.
  - The *miss penalty* is the time required to process a miss, including the time that it takes to replace a block of memory plus the time it takes to deliver the data to the processor.

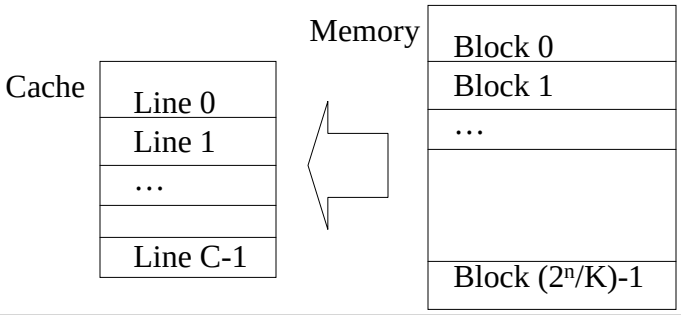
Cache Example

- Consider a Level 1 cache capable of holding 1000 words with a 0.1 μs access time. Level 2 is memory with a 1 μs access time.
- If 95% of memory access is in the cache:
  - T=(0.95)*(0.1 μs) + (0.05)*(0.1+1 μs) = 0.15 μs
- If 5% of memory access is in the cache:
  - T=(0.05)*(0.1 μs) + (0.95)*(0.1+1 μs) = 1.05 μs
- Want as many cache hits as possible!



Cache Design

- If memory contains 2^n addressable words
  - Memory can be broken up into blocks with K words per block. Number of blocks = 2^n / K
  - Cache consists of C **lines** or **slots**, each consisting of K words
  - C << M
  - How to map blocks of memory to lines in the cache?



Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Size does matter

- Cost
  - More cache is expensive
- Speed
  - More cache is faster
  - Up to a point - diminishing returns as cache increases in size
    - Also can take longer to search the more cache there is

Mapping Function

- Suppose we have the following configuration
  - Word size of 1 byte
  - Cache of 16 bytes
  - Cache line / Block size is 2 bytes
    - i.e. cache is 16/2 = 8 (2^3) lines of 2 bytes per line
    - Will need 8 addresses for a block in the cache
  - Main memory of 64 bytes
    - 6 bit address needed to reference 64 bytes
    - (2^6= 64)
    - 64 bytes / 2 bytes-per-block → 32 Memory Blocks
  - Somehow we have to map the 32 memory blocks to the 8 lines in the cache. Multiple memory blocks will have to map to the same line in the cache!

Mapping Function – 64K Cache Example

- Suppose we have the following configuration
  - Word size of 1 byte
  - Cache of 64 KByte
  - Cache line / Block size is 4 bytes
    - i.e. cache is 64 Kb / 4 bytes = 16,384 (2^{14}) lines of 4 bytes
  - Main memory of 16MBytes
    - 24 bit address
    - (2^{24}=16M)
    - 16Mb / 4bytes-per-block → 4 Meg of Memory Blocks
  - Somehow we have to map the 4 Meg of blocks in memory onto the 16K of lines in the cache. Multiple memory blocks will have to map to the same line in the cache!

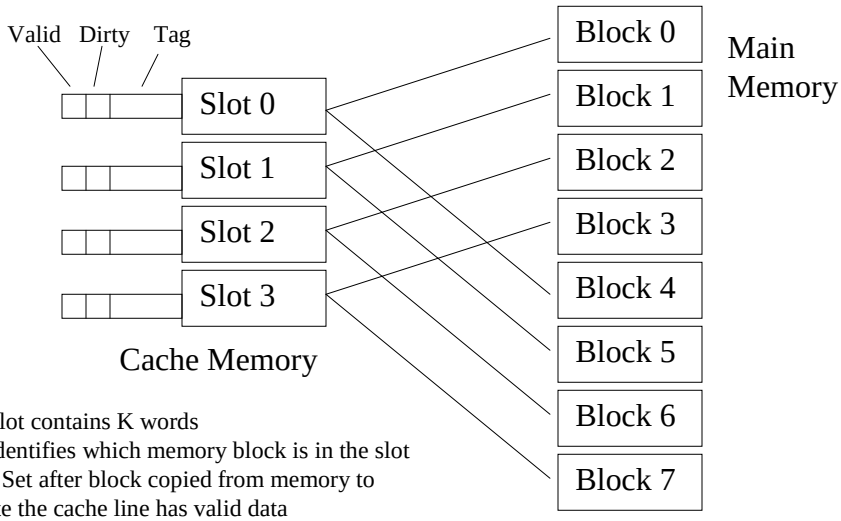
Direct Mapping

- Simplest mapping technique - 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
- Formula to map a memory block to a cache line:
  - i = j mod c
    - i=Cache Line Number
    - j=Main Memory Block Number
    - c=Number of Lines in Cache
  - i.e. we divide the memory block by the number of cache lines and the remainder is the cache line address

Direct Mapping with C=4

- Shrinking our example to a cache line size of 4 slots (each slot/line/block still contains 4 words):
  - Cache Line — Memory Block Held
    - 0 — 0, 4, 8, …
    - 1 — 1, 5, 9, …
    - 2 — 2, 6, 10, …
    - 3 — 3, 7, 11, …
  - In general:
    - 0 — 0, C, 2C, 3C, …
    - 1 — 1, C+1, 2C+1, 3C+1, …
    - 2 — 2, C+2, 2C+2, 3C+2, …
    - 3 — 3, C+3, 2C+3, 3C+3, …

Direct Mapping with C=4



Each slot contains K words
Tag: Identifies which memory block is in the slot
Valid: Set after block copied from memory to indicate the cache line has valid data

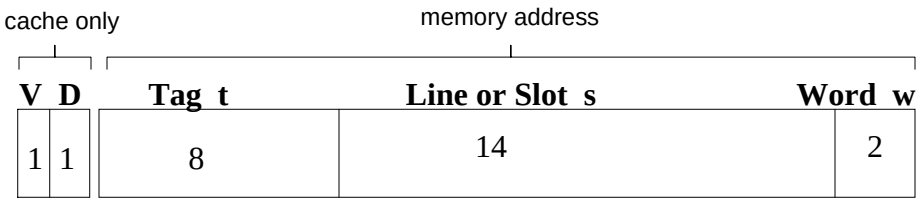
Direct Mapping Address Structure

- There is an easy way to compute
  i = j mod c
  based upon the bits in the address we are trying to access
- Address is in three parts
  - Least Significant w bits identify unique word within a cache line
    - w must be enough bits to address a specific word in a cache line; e.g. if 4 words per cache line, then we need 2 bits for w
  - Next Significant s bits specify which line this address maps into
    - s must be enough bits to address a specific line in the cache; e.g. if 16 cache lines, then we need 4 bits for s
  - Remaining t bits used as a tag to identify the memory block

Tag t	**Line or Slot s**	**Word w**
2	4	2

8 bit address

Direct Mapping Address 64K Cache Example



- Given a 24 bit address (to access 16Mb)
- 2 bit word identifier (4 byte block)
- Need 14 bits to address the cache slot/line
- Leaves 8 bits left for tag (=22-14)

- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag
- Also need a Valid bit and a Dirty bit
  - Valid – Indicates if the slot holds a block belonging to the program being executed
  - Dirty – Indicates if a block has been modified while in the cache. Will need to be written back to memory before slot is reused for another block

Direct Mapping Example, 64K Cache

Cache Memory

Addr	Tag	W0	W1	W2	W3
0	00	F1	F2	F3	F4
1	1B	11	12	13	14
2					
3					
4					
5					
..					
..					
2^{14}-1					

Main Memory

	Addr (hex)	Data
Line 0	000000	F1
	000001	F2
	000002	F3
	000003	F4
Line 1	000004	AB
	…	
Line 1	1B0004	11
	1B0005	12
	1B0006	13
	1B0007	14

1B0007 = 0001 1011 0000 0000 0000 0111
Word = 11, Line = 0000 0000 0000 01, Tag= 0001 1011

Cache Example

- The website for the textbook includes a link to CAMERA, a cache simulator
- Example for direct-mapped cache

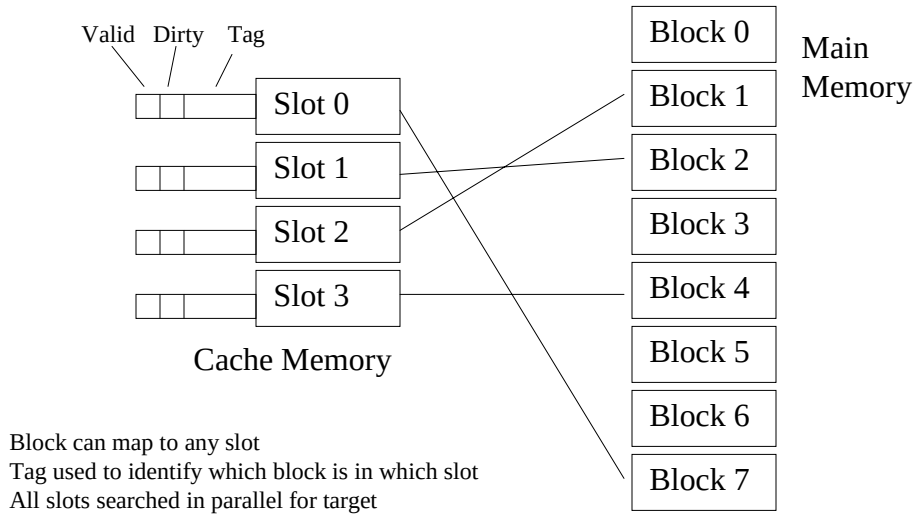
Direct Mapping pros & cons

- 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 – condition called **thrashing**

Fully Associative Mapping

- A fully associative mapping scheme can overcome the problems of the direct mapping scheme
  - 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
  - Also need a Dirty and Valid bit
- But Cache searching gets expensive!
  - Ideally need circuitry that can simultaneously examine all tags for a match
  - Lots of circuitry needed, high cost
- Need replacement policies now that anything can get thrown out of the cache (will look at this shortly)

Associative Mapping Example



Block can map to any slot
Tag used to identify which block is in which slot
All slots searched in parallel for target

Associative Mapping 64K Cache Example

Tag 22 bit	Word 2 bit

- 22 bit tag stored with each slot in the cache – no more bits for the slot line number needed since all tags searched in parallel
- Compare tag field of a target memory address with tag entry in cache to check for hit
- Least significant 2 bits of address identify which word is required from the block, e.g.:
  - Address: FFFFFC = 1111 1111 1111 1111 1111 1100
    - Tag: Left 22 bits, truncate on left:
      - 11 1111 1111 1111 1111 1111
      - 3FFFFF
  - Address: 16339C = 0001 0110 0011 0011 1001 1100
    - Tag: Left 22 bits, truncate on left:
      - 00 0101 1000 1100 1110 0111
      - 058CE7

CAMERA Example

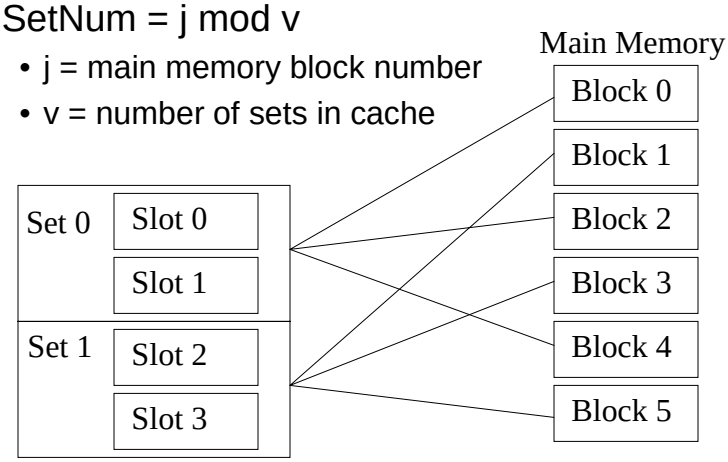
- CAMERA simulator for a fully-associative cache

Set Associative Mapping

- Compromise between fully-associative and direct-mapped cache
  - Cache is divided into a number of sets
  - Each set contains a number of lines
  - A given block maps to any line in a specific set
    - Use direct-mapping to determine which set in the cache corresponds to a set in memory
    - Memory block could then be in any line of that set
  - e.g. 2 lines per set
    - 2 way associative mapping
    - A given block can be in either of 2 lines in a specific set
  - e.g. K lines per set
    - K way associative mapping
    - A given block can be in one of K lines in a specific set
    - Much easier to simultaneously search one set than all lines

Set Associative Mapping

- To compute cache set number:
  - SetNum = j mod v
    - j = main memory block number
    - v = number of sets in cache

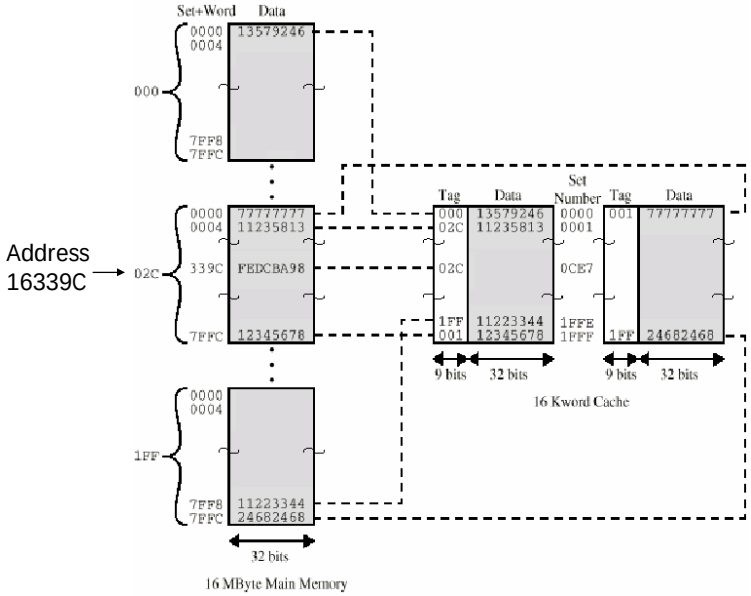


Set Associative Mapping 64K Cache Example

Tag 9 bit	Set 13 bit	Word 2 bit

- E.g. Given our 64Kb cache, with a line size of 4 bytes, we have 16384 lines. Say that we decide to create 8192 sets, where each set contains 2 lines. Then we need 13 bits to identify a set (2^{13}=8192)
- Use set field to determine cache set to look in
- Compare tag field of all slots in the set to see if we have a hit, e.g.:
  - Address = 16339C = 0001 0110 0011 0011 1001 1100
    - Tag = 0 0010 1100 = 02C
    - Set = 0 1100 1110 0111 = 0CE7
    - Word = 00 = 0
  - Address = 008004 = 0000 0000 1000 0000 0000 0100
    - Tag = 0 0000 0001 = 001
    - Set = 0 0000 0000 0001 = 0001
    - Word = 00 = 0

Two Way Set Associative Example



CAMERA Example

- CAMERA simulator for a k-way set associative cache

K-Way Set Associative

- Two-way set associative gives much better performance than direct mapping
  - Just one extra slot avoids the thrashing problem
- Four-way set associative gives only slightly better performance over two-way
- Further increases in the size of the set has little effect other than increased cost of the hardware!

Replacement Policy (1)

- The replacement policy is the technique we use to determine which line in the cache should be thrown out when we want to put a new block in from memory
- Direct mapping
  - No choice
  - Each block only maps to one line
  - Replace that line

Replacement Algorithms (2) Associative & Set Associative

- Algorithm must be implemented in hardware (speed)
- Least Recently used (LRU)
  - e.g. in 2 way set associative, which of the 2 block is LRU?
    - For each slot, have an extra bit, USE. Set to 1 when accessed, set all others to 0.
  - For more than 2-way set associative, need a time stamp for each slot - expensive
- First in first out (FIFO)
  - Replace block that has been in cache longest
  - Easy to implement as a circular buffer
- Least frequently used
  - Replace block which has had fewest hits
  - Need a counter to sum number of hits
- Random
  - Almost as good as LFU and simple to implement

Write Policy

- So far we've only discussed reading memory, we may also write back to memory
- If a memory write only updates the cache, then the cache is now inconsistent with main memory
  - Could cause problems
  - Reading in another memory block that maps to the same cache line
  - I/O device or another processor might directly try to read/write to memory

Write through

- Simplest technique to handle the cache inconsistency problem - All writes go to main memory as well as cache.
- Multiple CPUs must monitor main memory traffic (snooping) to keep local cache local to its CPU up to date in case another CPU also has a copy of a shared memory location in its cache
- Simple but Lots of traffic
- Slows down writes

Write Back

- Updates initially made in cache only
  - Dirty bit is set when we write to the cache, this indicates the cache is now inconsistent with main memory
- Dirty bit for cache slot is cleared when update occurs
- If cache line is to be replaced, write the existing cache line to main memory if dirty bit is set before loading the new memory block

Cache Performance

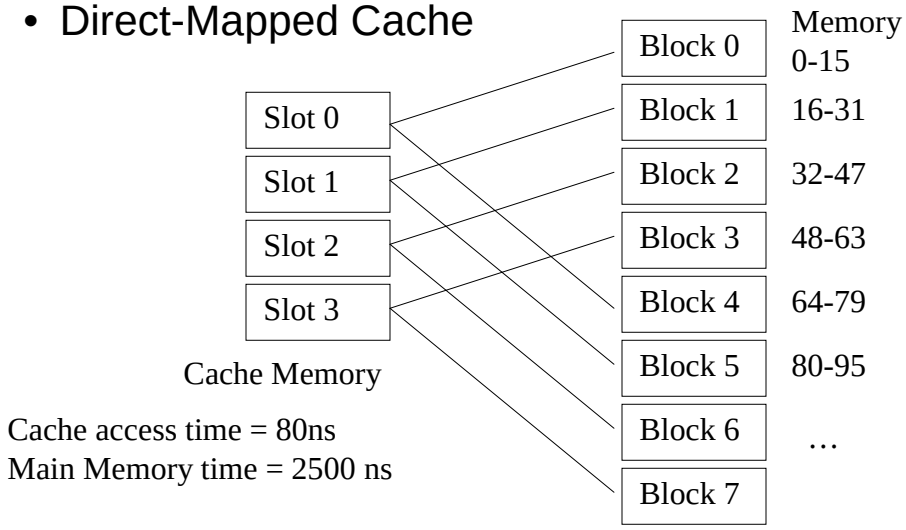
- Two measures that characterize the performance of a cache are the **hit ratio** and the **effective access time**

$$\text{Hit Ratio} = \frac{\text{(Num times referenced words are in cache)}}{\text{(Total number of memory accesses)}}$$

$$\text{Eff. Access Time} = \frac{\text{(\# hits)(TimePerHit)+(\# misses) (TimePerMiss)}}{\text{(Total number of memory accesses)}}$$

Cache Performance Example

- Direct-Mapped Cache

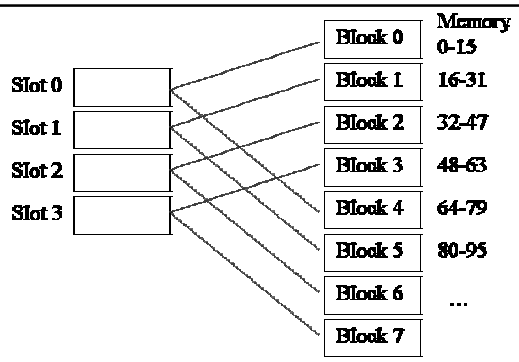


Cache access time = 80ns
Main Memory time = 2500 ns

Cache Performance Example

- Sample program executes from memory location 48-95 once. Then it executes from 15-31 in a loop ten times before exiting.

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213		Total misses = 5	



Once:
48-95

Ten Times:
15-31

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213		Total misses = 5	

Cache Performance Example

- Hit Ratio: 213 / 218 = 97.7%
- Effective Access Time:
  ((213)*(80ns)+(5)(2500ns)) / 218 = 136 ns

- Although the hit ratio is high, the effective access time in this example is 75% longer than the cache access time due to the large amount of time spent during a cache miss
- What sequence of main memory block accesses would result in much worse performance?