AAiT

Computer Architecture & Organization

Chapter 11

Cache Memory

Key Characteristics of Computer Memory Systems

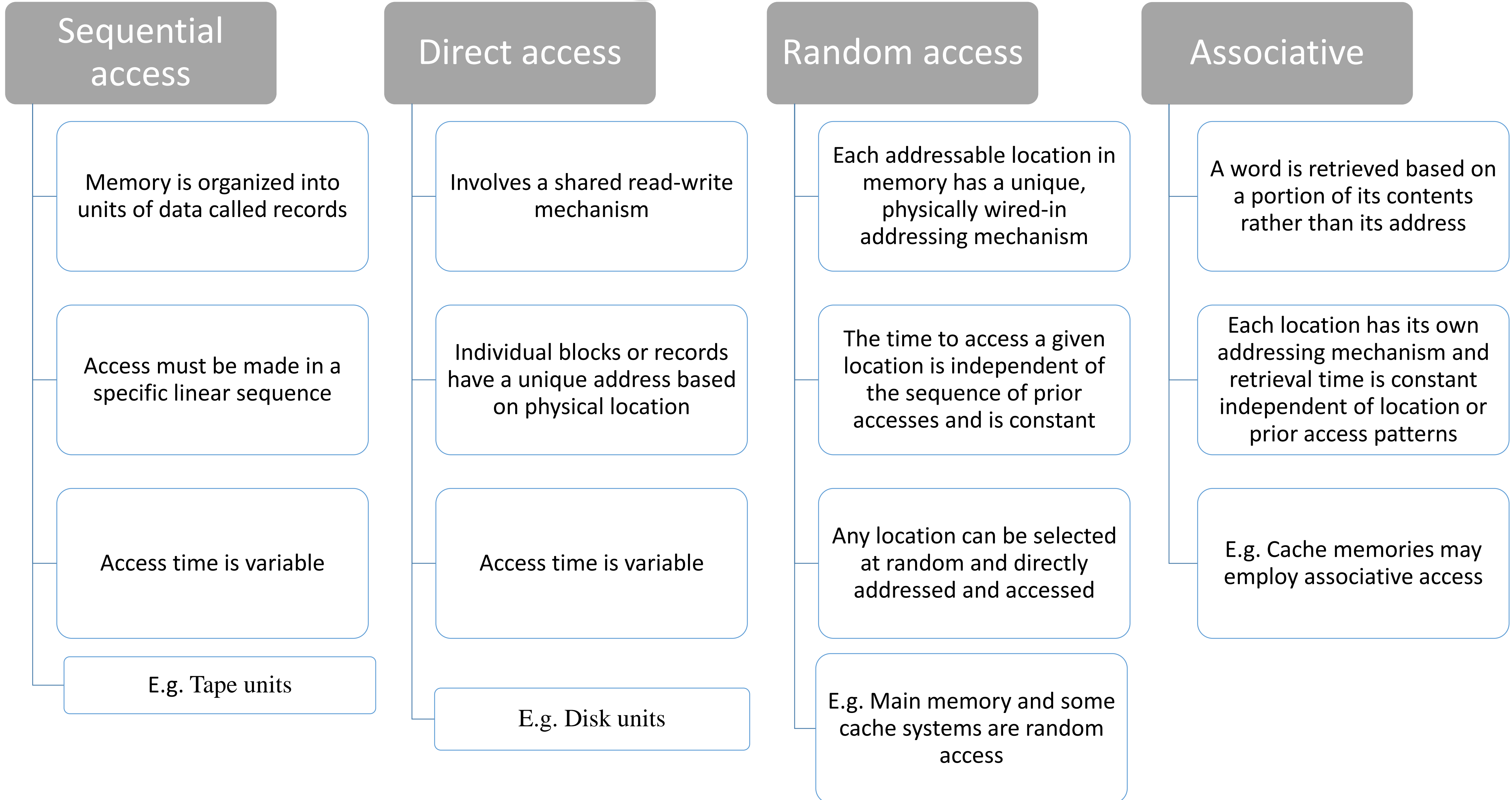
Location	Performance
Internal (e.g. processor registers, cache, main memory)	Access time
External (e.g. optical disks, magnetic disks, tapes)	Cycle time
Capacity	Transfer rate
Number of words	**Physical Type**
Number of bytes	Semiconductor
Unit of Transfer	Magnetic
Word	Optical
Block	Magneto-optical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

Table 4.1 Key Characteristics of Computer Memory Systems

Characteristics of Memory Systems

- Location
  - Refers to whether memory is internal and external to the computer
  - Internal memory is often equated with main memory
  - Processor requires its own local memory, in the form of registers
  - Cache is another form of internal memory
  - External memory consists of peripheral storage devices that are accessible to the processor via I/O controllers
- Capacity
  - Memory is typically expressed in terms of bytes
- Addressable units
  - In some systems, the addressable unit is the word. However, many systems allow addressing at the byte level. In any case, the relationship between the length in bits A of an address and the number N of addressable units is $2^A = N$.
- Unit of transfer
  - For internal memory the unit of transfer is equal to the number of electrical lines into and out of the memory module. The unit of transfer need not equal a word or an addressable unit. For external memory, data are often transferred in much larger units than a word, and these are referred to as blocks

Method of Accessing Units of Data



Capacity and Performance:

Capacity and performance are the two most important characteristics of memory

Three performance parameters are used:

Access time (latency)

- For random-access memory it is the time it takes to perform a read or write operation
- For non-random-access memory it is the time it takes to position the read-write mechanism at the desired location

Memory cycle time

- Access time plus any additional time required before second access can commence
- Additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively
- Concerned with the system bus, not the processor

Transfer rate

- The rate at which data can be transferred into or out of a memory unit
- For random-access memory it is equal to 1/(cycle time)

Memory

- The most common forms are:
  - Semiconductor memory
  - Magnetic surface memory
  - Optical
  - Magneto-optical
- Several physical characteristics of data storage are important:
  - Volatile memory
    - Information decays naturally or is lost when electrical power is switched off
  - Nonvolatile memory
    - Once recorded, information 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
  - Nonerasable memory
    - Cannot be altered, except by destroying the storage unit
    - Semiconductor memory of this type is known as read-only memory (ROM)
- For random-access memory the organization is a key design issue
  - Organization refers to the physical arrangement of bits to form words

AAiT

Memory Hierarchy

- Design constraints on a computer's memory can be summed up by three questions:
  - How much, how fast, how expensive
- There is a trade-off among 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 the memory dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy

Memory Hierarchy - Diagram

A typical hierarchy is illustrated in Figure 4.1. As one goes down the hierarchy, the following occur:

a. Decreasing cost per bit

b. Increasing capacity

c. Increasing access time

d. Decreasing frequency of access of the memory by the processor

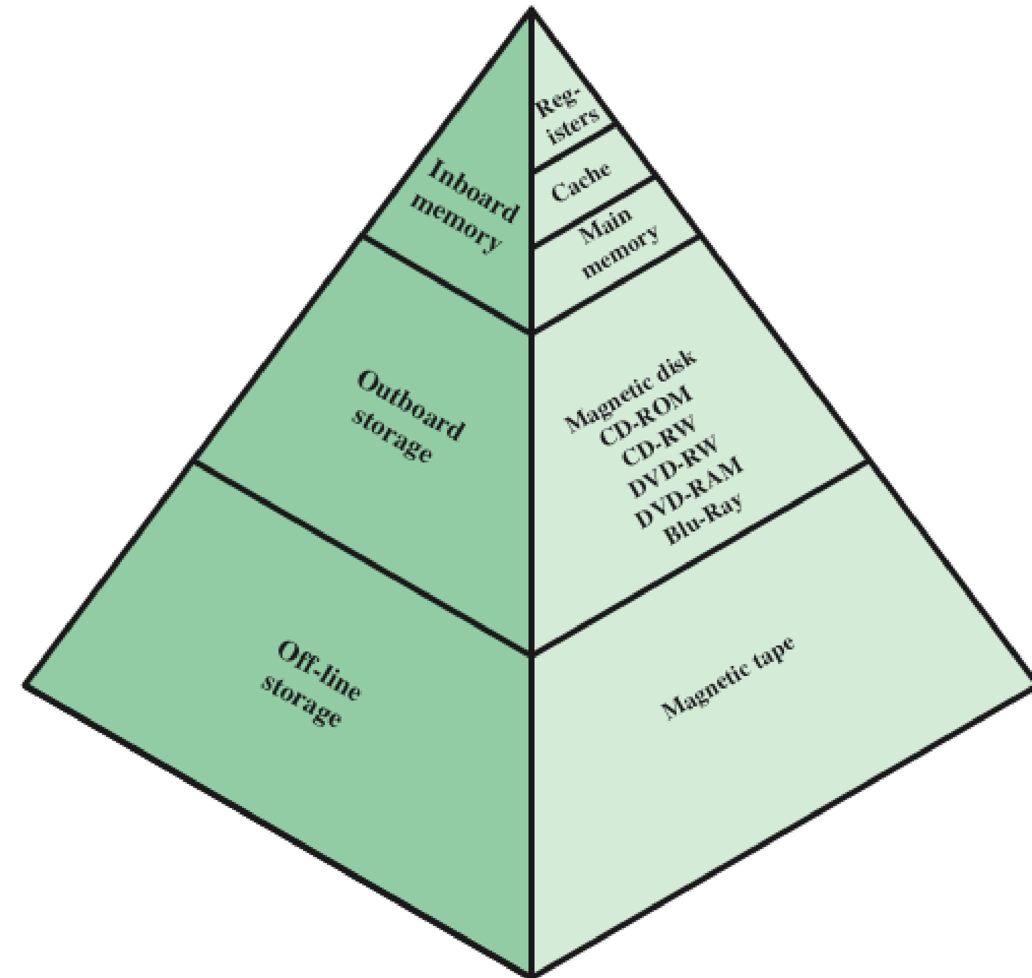
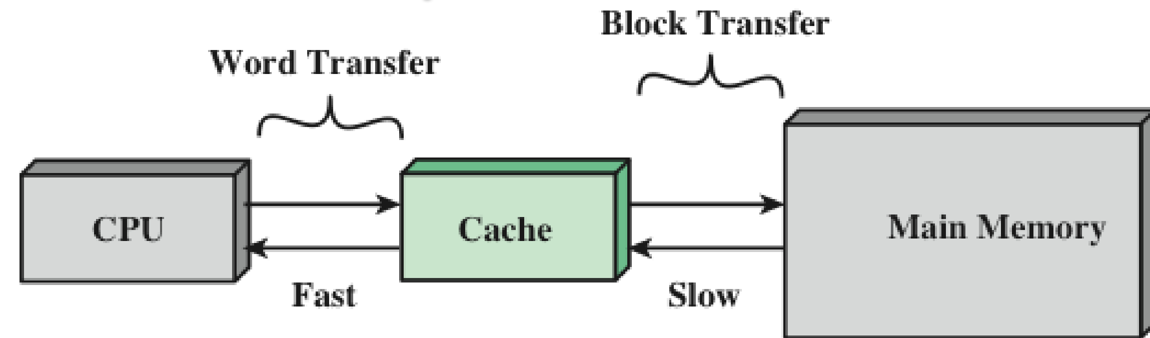


Figure 4.1 The Memory Hierarchy

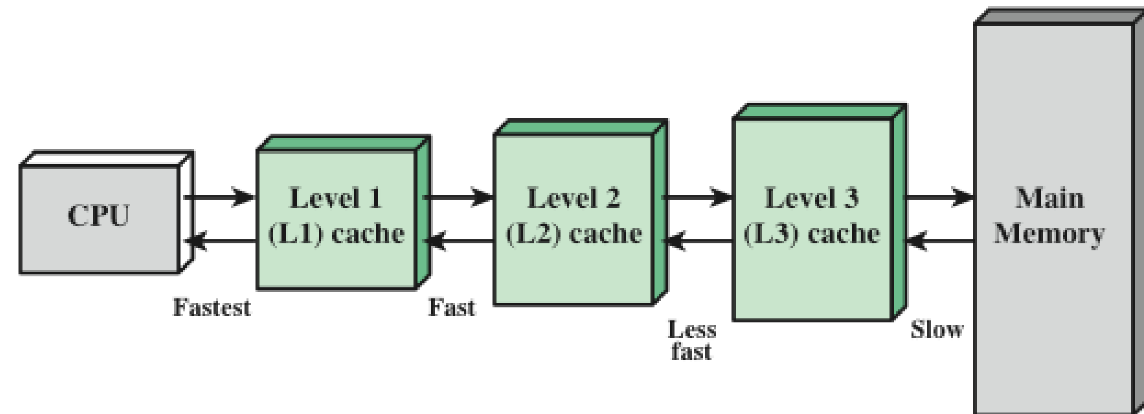
Cache and Main Memory

- If the word the processors is looking for is not in the cache, a block of main memory is read into the cache. Because of the phenomenon of locality of reference, 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.

- Figure 4.3b depicts the use of multiple levels of cache. The L2 cache is slower and typically larger than the L1 cache, and the L3 cache is slower and typically larger than the L2 cache.



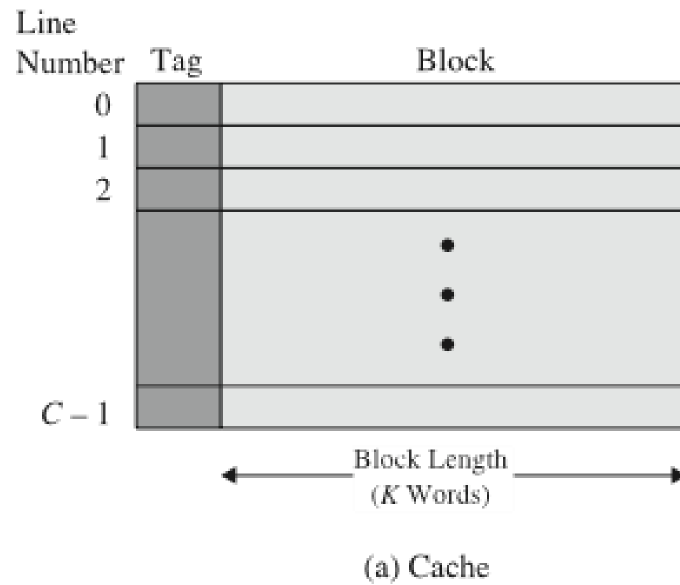
(a) Single cache



(b) Three-level cache organization

Figure 4.3 Cache and Main Memory

Cache/Main Memory Structure



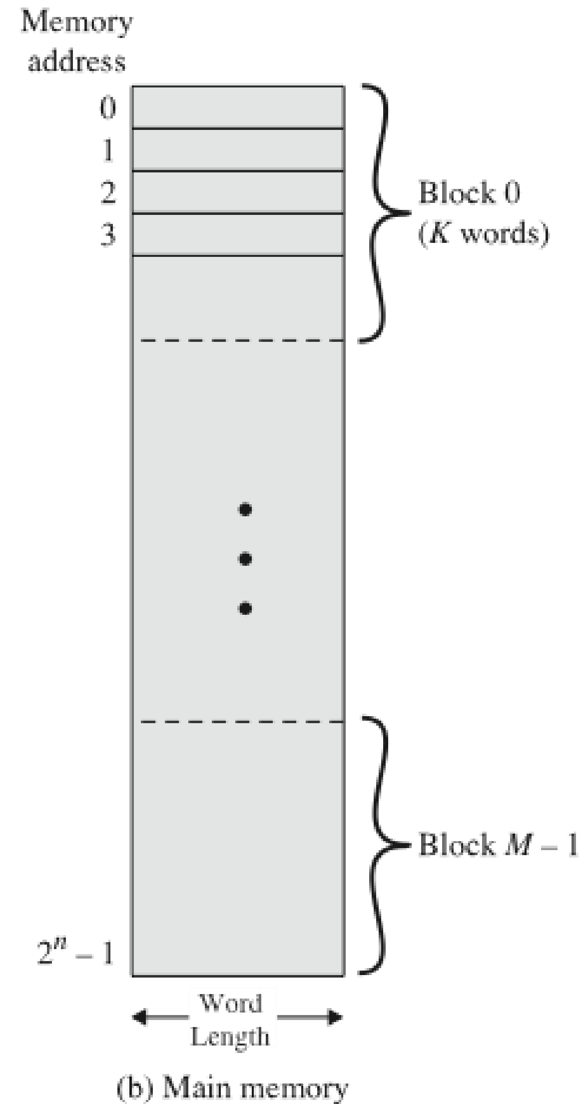


Figure 4.4 Cache/Main-Memory Structure

AAiT

Cache Read Operation

Figure 4.5 illustrates the read operation. The processor generates the read address (RA) of a word to be read. If the word is contained in the cache, it is delivered to the processor. Otherwise, the block containing that word is loaded into the cache, and the word is delivered to the processor.

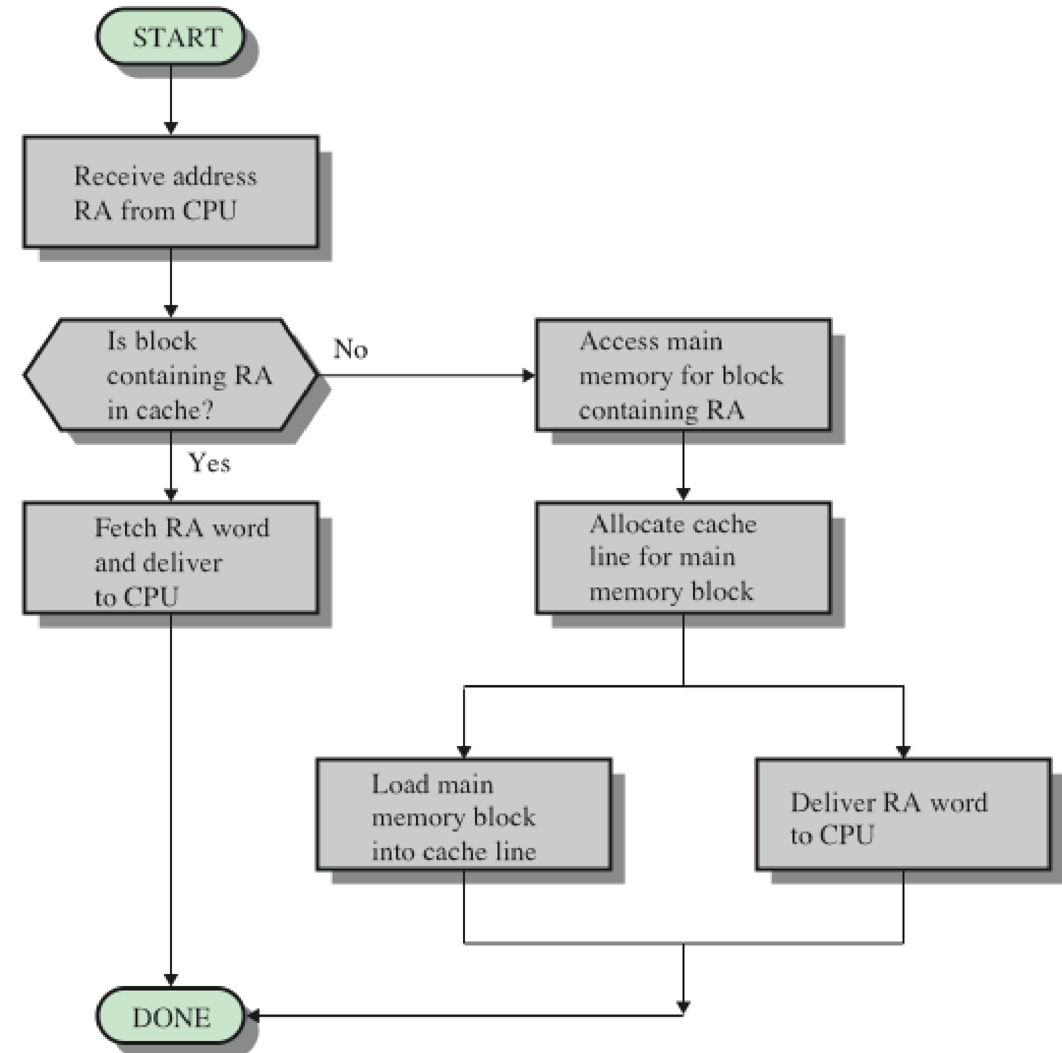


Figure 4.5 Cache Read Operation

Typical Cache Organization

When a cache hit occurs, the data and address buffers are disabled and communication is only between processor and cache, with no system bus traffic. When a cache miss occurs, the desired address is loaded onto the system bus and the data are returned through the data buffer to both the cache and the processor.

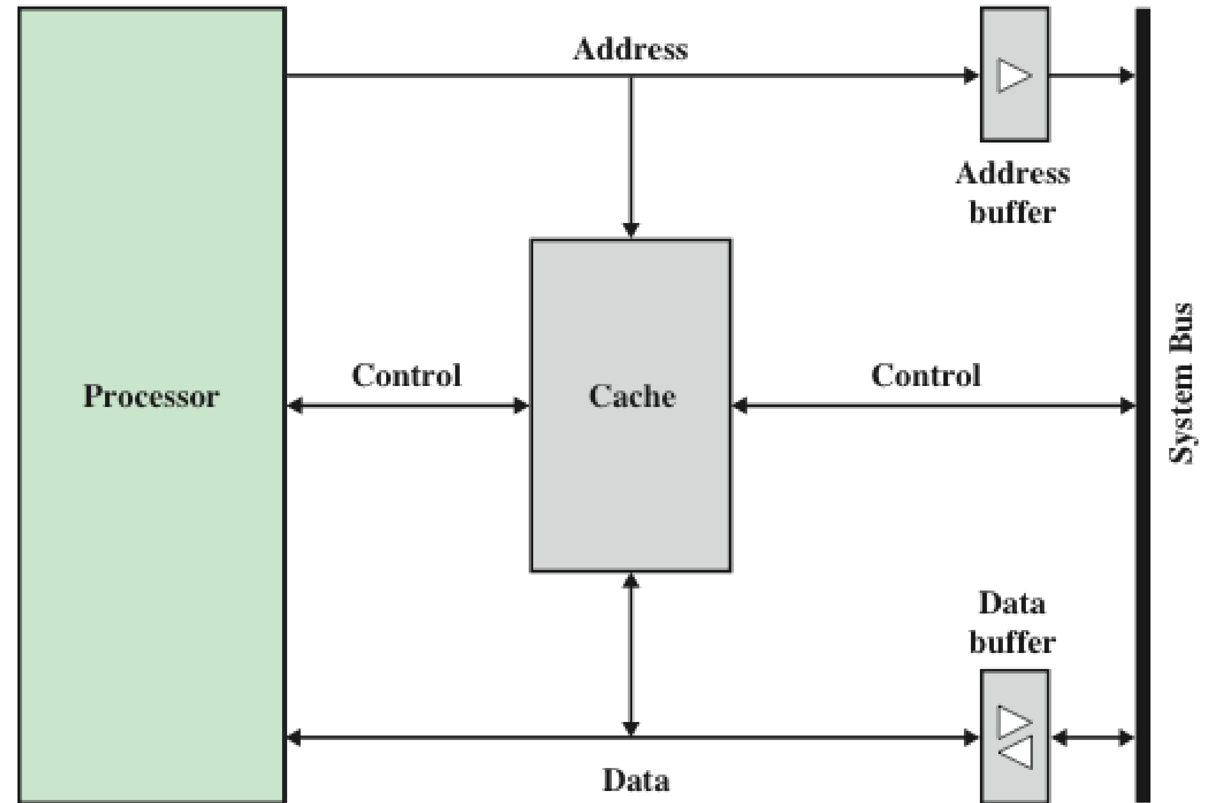


Figure 4.6 Typical Cache Organization

Elements of Cache Design

Although there are a large number of cache implementations, there are a few basic design elements that serve to classify and differentiate cache architectures. Table 4.2 lists key elements.

Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back

Line Size

Number of Caches
- Single or two level
- Unified or split

Table 4.2 Elements of Cache Design

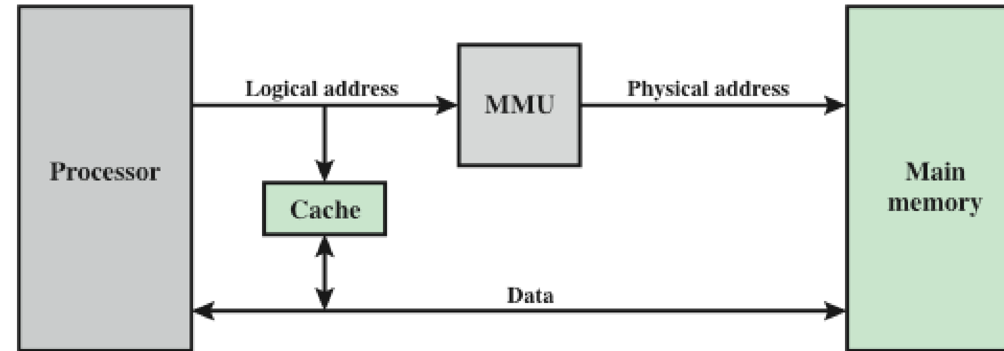
AAiT

Cache Addresses

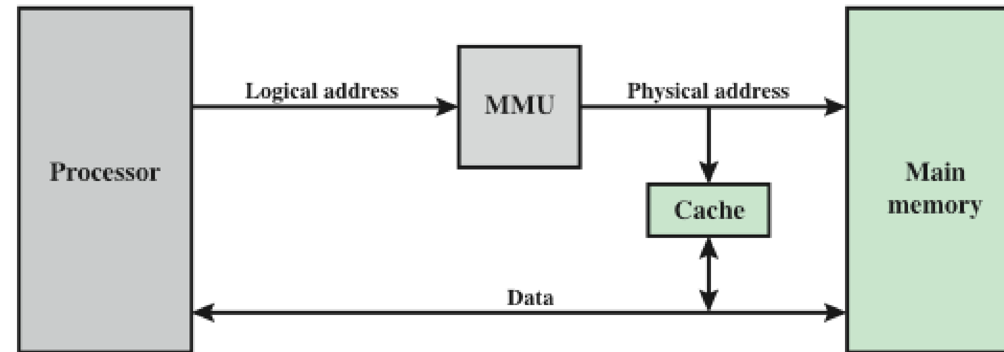
- Virtual memory
  - Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
  - When used, the address fields of machine instructions contain virtual addresses
  - For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory

AAiT

Logical and Physical Caches



(a) Logical Cache



(b) Physical Cache

Figure 4.7 Logical and Physical Caches

AAiT

Table 4.3

Cache Sizes of Some Processors

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 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
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	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24-48 MB
Intel Core i7 EE 990	Workstaton/ server	2011	6 × 32 kB/32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ Server	2011	24 × 64 kB/ 128 kB	24 × 1.5 MB	24 MB L3 192 MB L4

[a] Two values separated by a slash refer to instruction and data caches.

[b] Both caches are instruction only; no data caches.

Mapping Function

- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines
- Three techniques can be used:

Direct

- The simplest technique
- Maps each block of main memory into only one possible cache line

Associative

- Permits each main memory block to be loaded into any line of the cache
- The cache control logic interprets a memory address simply as a Tag and a Word field
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match

Set Associative

- A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages

Example 4.2 For all three cases, the example includes the following elements:

- The cache can hold 64 Kbytes.
- Data are transferred between main memory and the cache in blocks of 4 bytes each. This means that the cache is organized as ______ lines of 4 bytes each.
- The main memory consists of 16 Mbytes, with each byte directly addressable by a _ bit address ($2^{24} = 16M$). Thus, for mapping purposes, we can consider main memory to consist of 4M blocks of 4 bytes each.

Solution:

- Data are transferred between main memory and the cache in blocks of 4 bytes each. This means that the cache is organized as $16K = 2^{14}$ lines of 4 bytes each.
- The main memory consists of 16 Mbytes, with each byte directly addressable by a 24-bit address ($2^{24} = 16M$). Thus, for mapping purposes, we can consider main memory to consist of 4M blocks of 4 bytes each.

Direct Mapping

The mapping is expressed as

$$i = j \text{ modulo } m$$

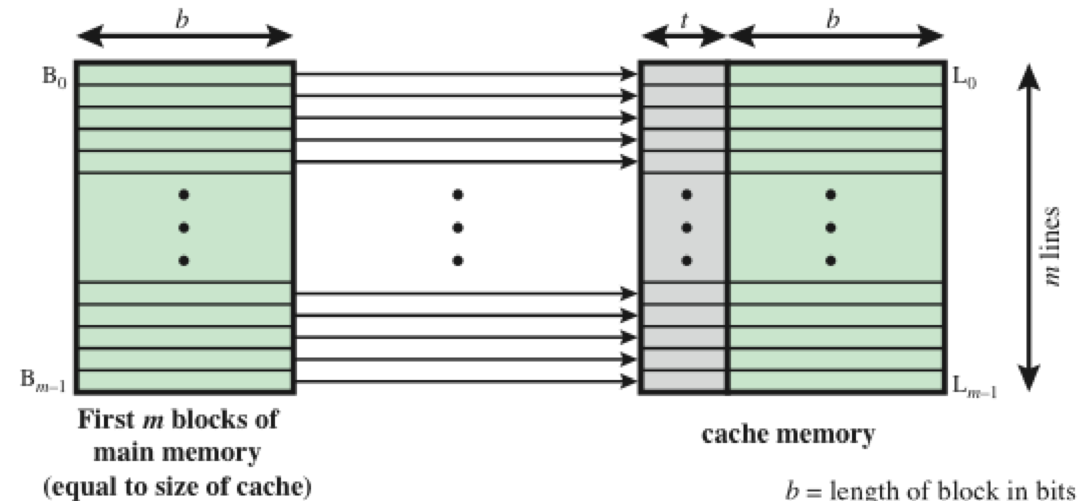
where

i = *cache line number*

j = *main memory block number*

m = *number of lines in the cache*

Figure 4.8a shows the mapping for the first *m blocks of main memory. Each* block of main memory maps into one unique line of the cache. The next *m blocks* of main memory map into the cache in the same fashion; that is, block B_m *of main* memory maps into line L_0 of cache, block B_{m+1} *maps into line* L_1, *and so on.*



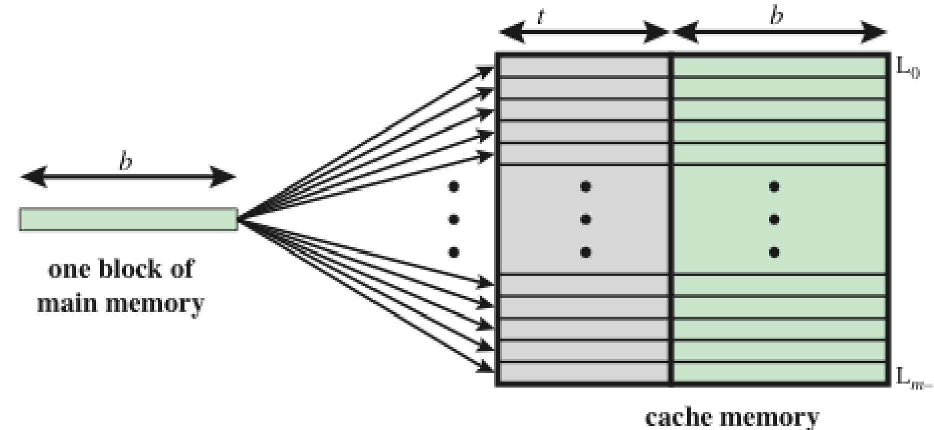


Figure 4.8 Mapping From Main Memory to Cache: Direct and Associative

Direct Mapping Cache Organization

AAiT

The mapping function is easily implemented using the main memory address.

Figure 4.9 illustrates the general mechanism.

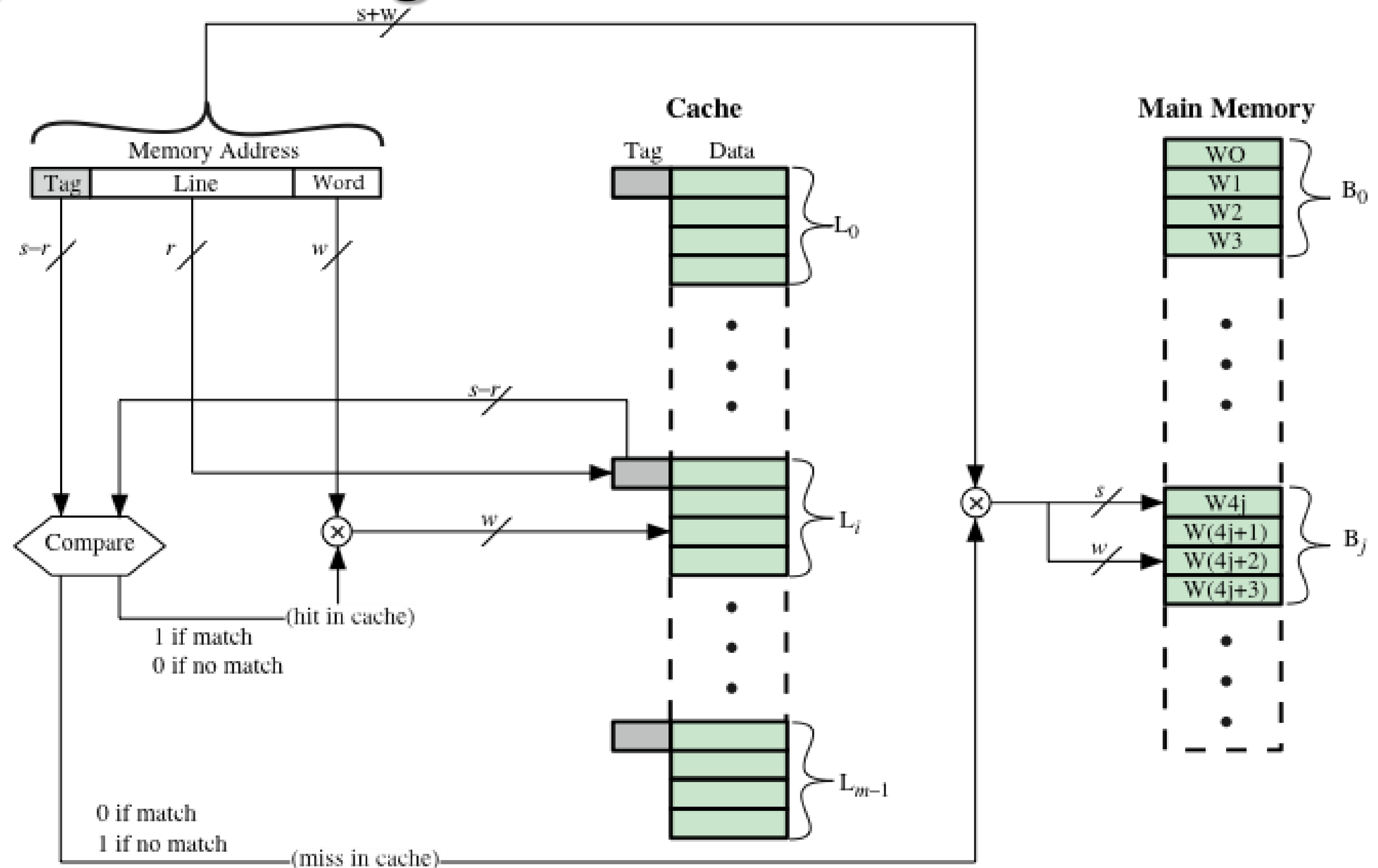


Figure 4.9 Direct-Mapping Cache Organization

AAiT

Example 4.2a Figure 4.10 shows our example system using direct mapping.[5] In the example, $m = 16K = 2^{14}$ and $i = j$ modulo 2^{14}. The mapping becomes

Cache Line	Starting Memory Address of Block
0	000000, 010000, …, FF0000
1	000004, 010004, …, FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, …, FFFFFC

Note that no two blocks that map into the same line number have the same tag number. Thus, blocks with starting addresses 000000, 010000, …, FF0000 have tag numbers 00, 01, …, FF, respectively.

Referring back to Figure 4.5, a read operation works as follows. The cache system is presented with a 24-bit address. The 14-bit line number is used as an index into the cache to access a particular line. If the 8-bit tag number matches the tag number currently stored in that line, then the 2-bit word number is used to select one of the 4 bytes in that line. Otherwise, the 22-bit tag-plus-line field is used to fetch a block from main memory. The actual address that is used for the fetch is the 22-bit tag-plus-line concatenated with two 0 bits, so that 4 bytes are fetched starting on a block boundary.

[5]In this and subsequent figures, memory values are represented in hexadecimal notation. See Chapter 9 for a basic refresher on number systems (decimal, binary, hexadecimal).

Homework

Do example 4.2a again for $m=4K=2^{12}$. Assuming the address bit length (s+w) remains 24 and Block size =4 byte:

a) Calculate starting Memory Address of Block for each cache line as shown in the example.

b) what will be bit size of Tag (s-r), Line (r), word (w) fields of the physical adrress?

c) Calculate number of addressable units on the main RAM, number of blocks in RAM and number of lines in cache

Direct Mapping Example

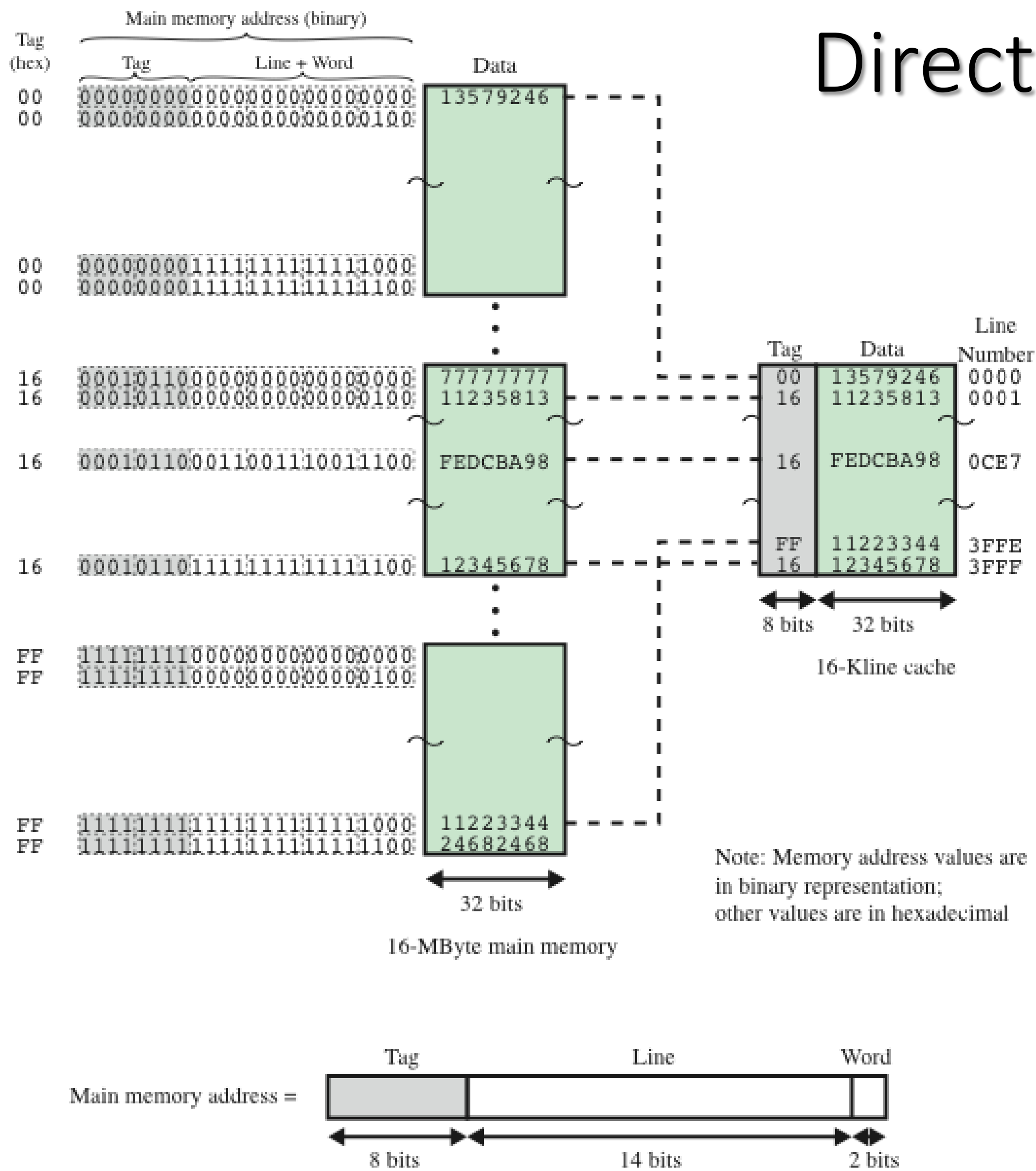


Figure 4.10 Direct Mapping Example

The effect of direct_mapping is that blocks of main memory are assigned to lines of the cache as follows:

Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
$\vdots$	$\vdots$
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

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

Victim Cache

- Main disadvantage of the direct mapping is that there is a fixed cache location for any given block. Thus, if a program happens to reference words repeatedly from two different blocks that map into the same cache line, then the blocks will be continually swapped in the cache, and the hit ratio will be low (a phenomenon known as thrashing).
- Victim Cache is originally proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time
- Victim cache is a fully associative cache, whose size is typically 4 to 16 cache lines, residing between a direct mapped L1 cache and the next level of memory.

Fully Associative Cache Organization

AAiT

- Associative mapping overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of the cache (Figure 4.8b).
- In this case, the cache control logic interprets a memory address simply as a Tag and a Word field.
- The Tag field uniquely identifies a block of main memory.
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's tag for a match.

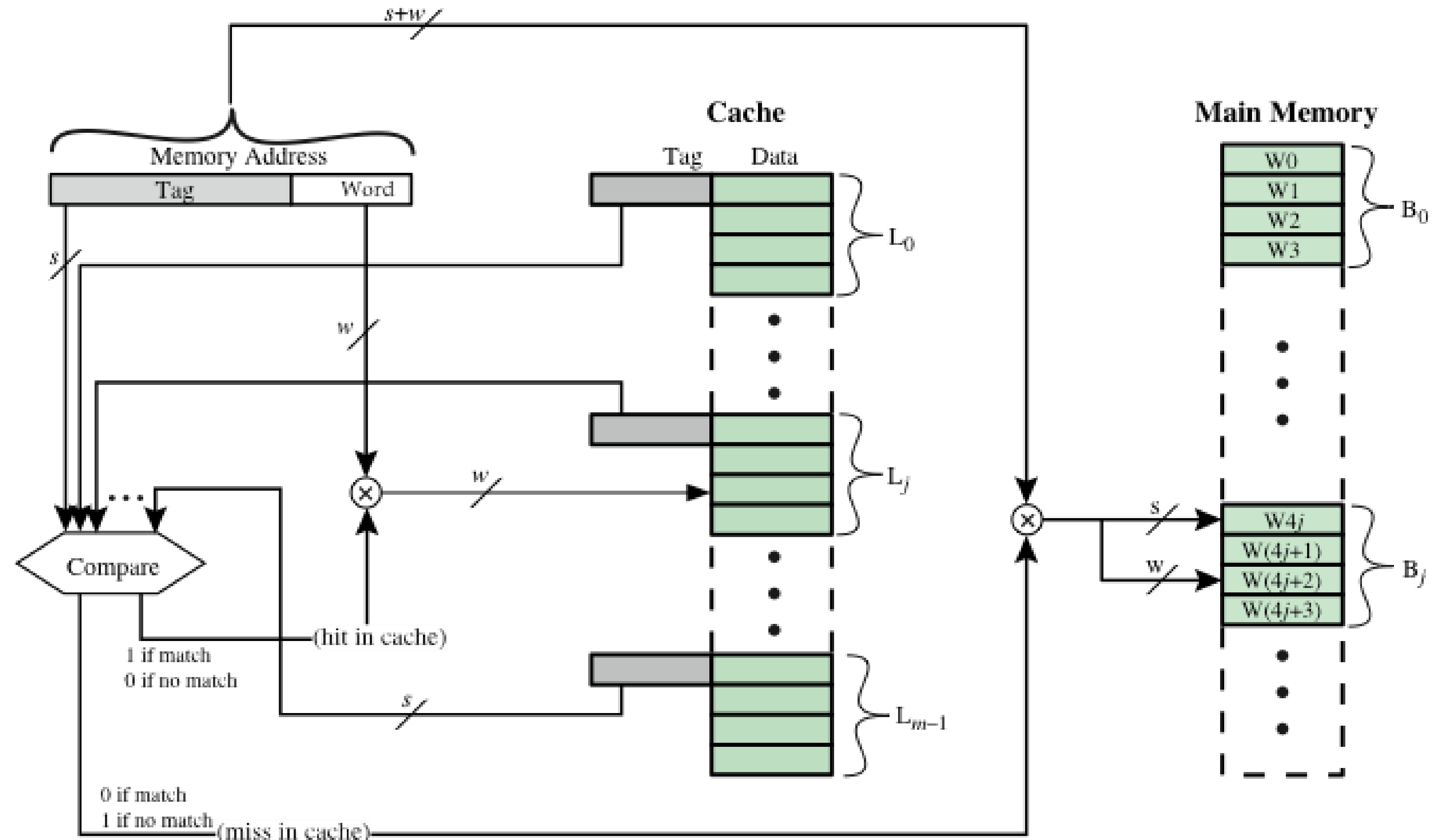


Figure 4.11 Fully Associative Cache Organization

Associative Mapping Example

Example 4.2b Figure 4.12 shows our example using associative mapping. A main memory address consists of a 22-bit tag and a 2-bit byte number. The 22-bit tag must be stored with the 32-bit block of data for each line in the cache. Note that it is the leftmost (most significant) 22 bits of the address that form the tag. Thus, the 24-bit hexadecimal address 16339C has the 22-bit tag 058CE7. This is easily seen in binary notation:

memory address	0001	0110	0011	0011	1001	1100	(binary)
	1	6	3	3	9	C	(hex)
tag (leftmost 22 bits)	00	0101	1000	1100	1110	0111	(binary)
	0	5	8	C	E	7	(hex)

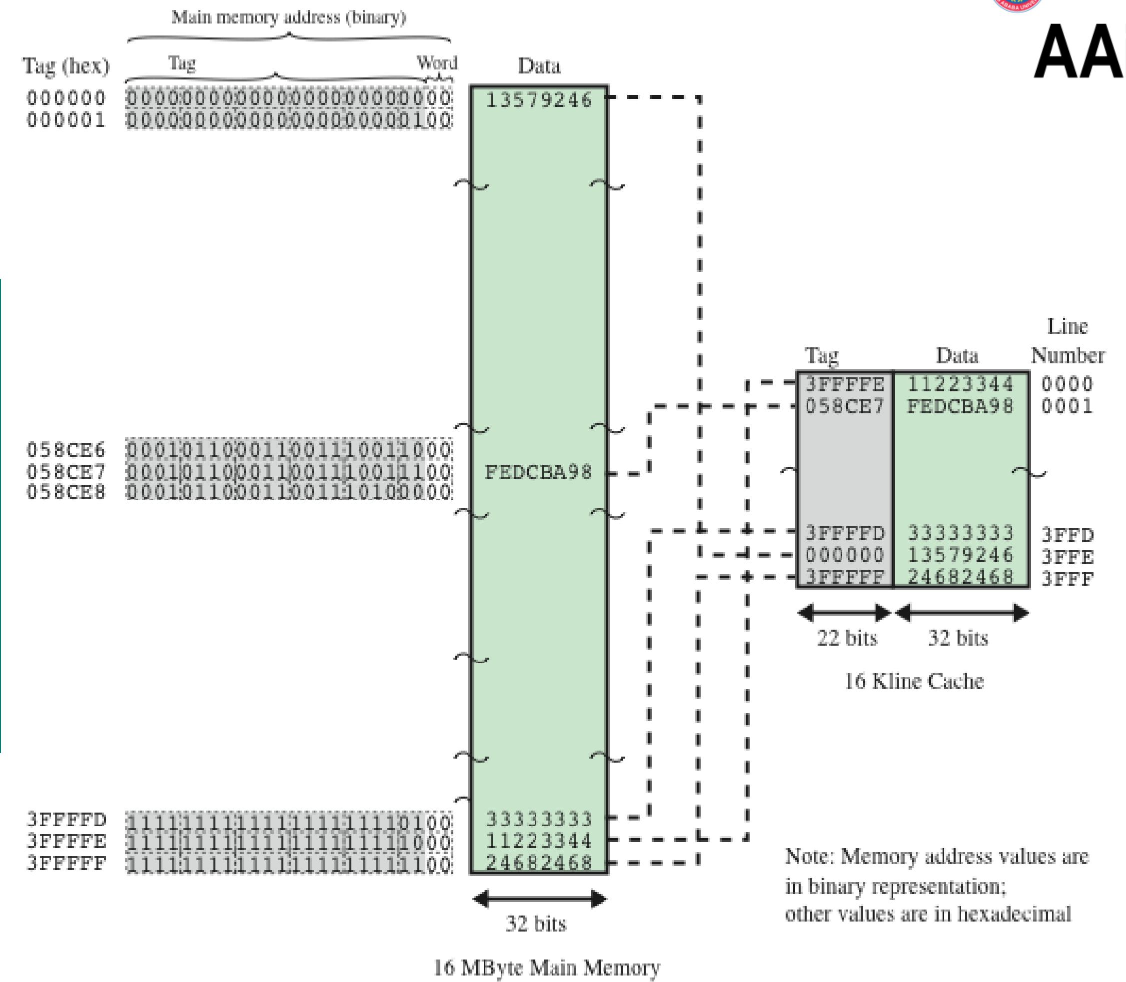


Figure 4.12 Associative Mapping Example

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

Cons and prones:

- With associative mapping, there is flexibility as to which block to replace when a new block is read into the cache. Replacement algorithms, discussed later in this section, are designed to maximize the hit ratio.
- The principal disadvantage of associative mapping is the complex circuitry required to examine the tags of all cache lines in parallel.

Set Associative Mapping

- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

Mapping From Main Memory to Cache:

k-Way Set Associative

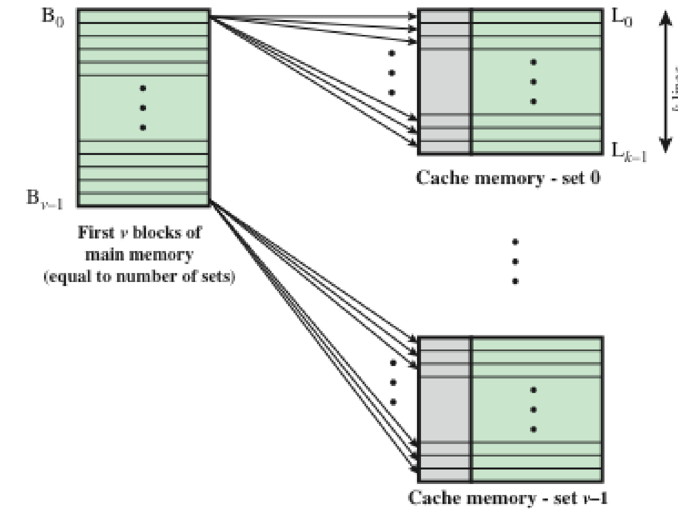
Figure 4.13a

- For the first v blocks of main memory , with set-associative mapping, each word maps into all the cache lines in a specific set out of v sets, so that main memory block B_0 maps into set 0, and so on.
- Thus, the set-associative cache can be physically implemented as <u>*n associative caches.*</u>

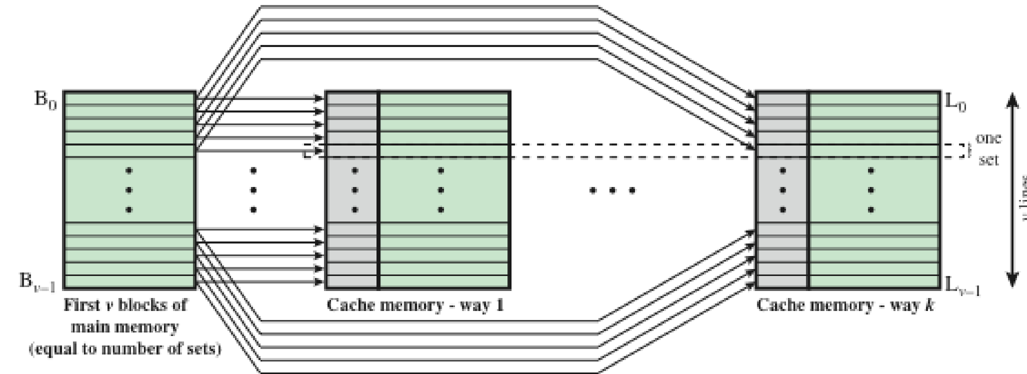
Figure 4.13a

- It is also possible to implement the set-associative cache as <u>*k direct-mapping caches*</u>.

AAiT



(a) v associative-mapped caches



(b) k direct-mapped caches

Figure 4.13 Mapping From Main Memory to Cache: *k*-way Set Associative

k-Way Set Associative Cache Organization

- Figure 4.14 illustrates the cache control logic.
- With fully associative mapping, the tag in a memory address is quite large and must be compared to the tag of every line in the cache.
- With *k-way set-associative mapping, the tag in a memory address is* much smaller and is only compared to the *k tags within a single set.*
- The *d set bits specify one of* $v = 2^d$ *sets.*

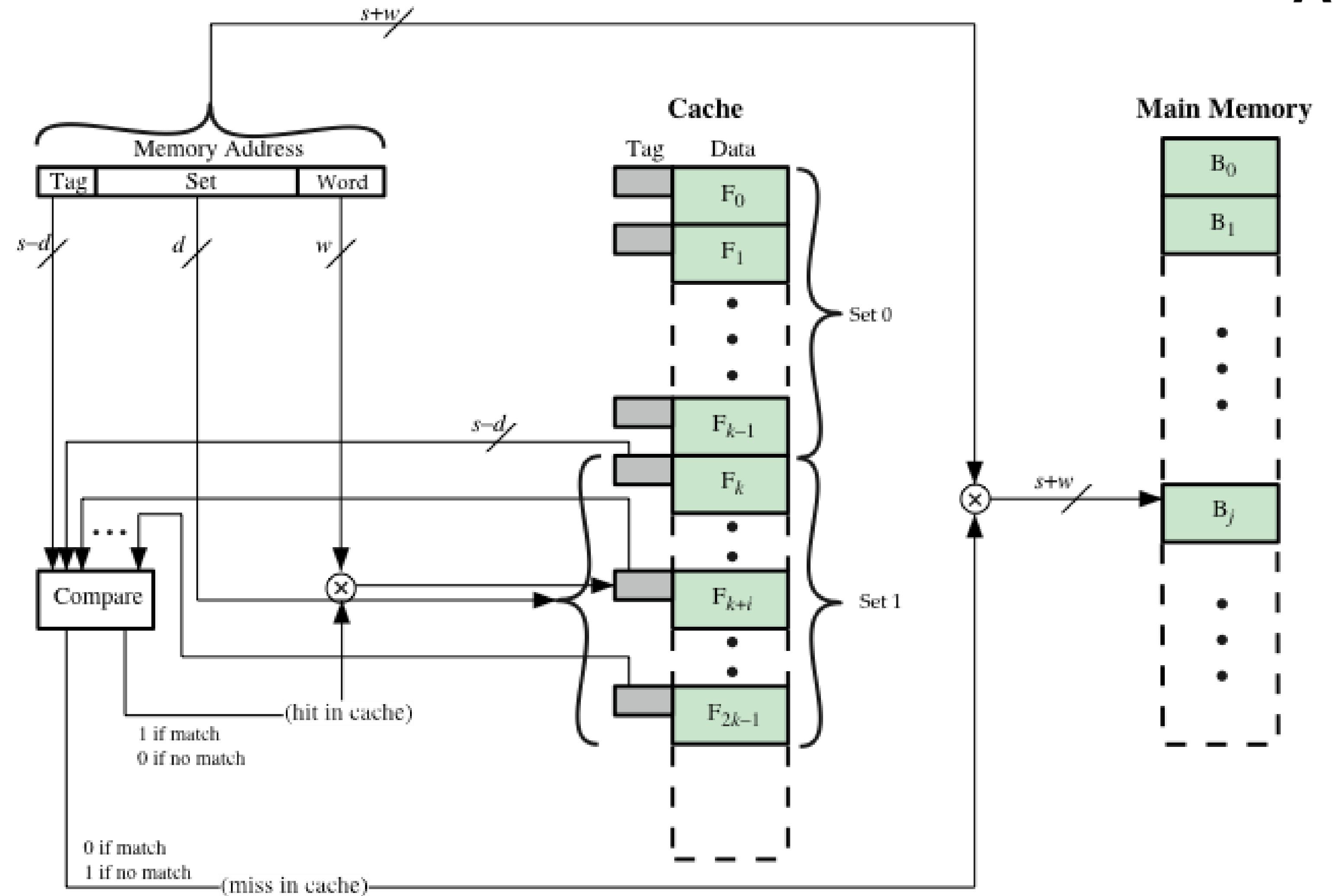


Figure 4.14 ***k*-Way Set Associative Cache Organization**

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 = m=kv = k * 2^d
- Size of cache = $k * 2^{d+w}$ words or bytes
- Size of tag = (s – d) bits

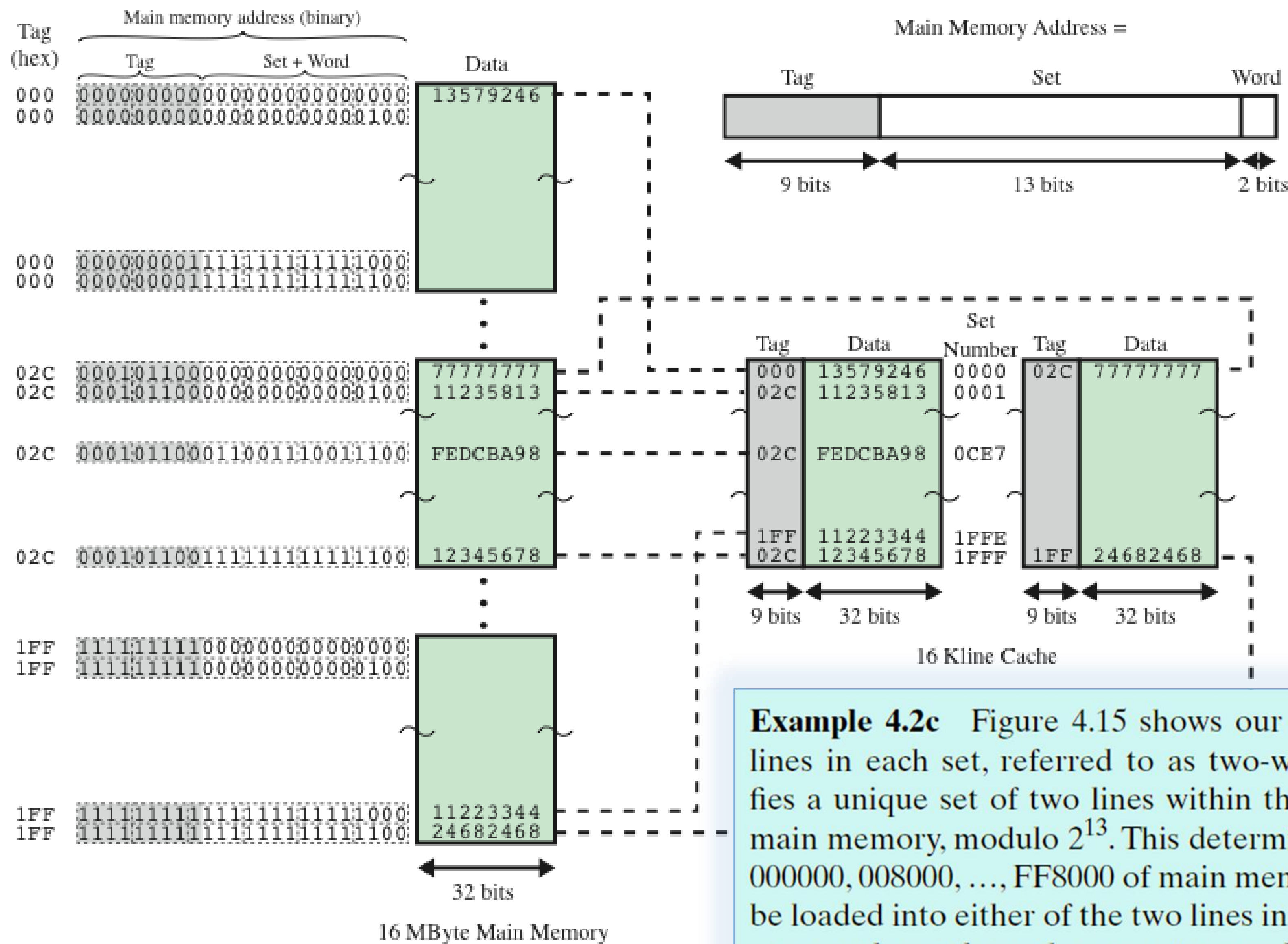


Figure 4.15 Two-Way Set Associative Mapping Example

Note: Memory address values are in binary representation; other values are in hexadecimal

Example 4.2c Figure 4.15 shows our example using set-associative mapping with two lines in each set, referred to as two-way set-associative. The 13-bit set number identifies a unique set of two lines within the cache. It also gives the number of the block in main memory, modulo 2^{13}. This determines the mapping of blocks into lines. Thus, blocks 000000, 008000, …, FF8000 of main memory map into cache set 0. Any of those blocks can be loaded into either of the two lines in the set. Note that no two blocks that map into the same cache set have the same tag number. For a read operation, the 13-bit set number is used to determine which set of two lines is to be examined. Both lines in the set are examined for a match with the tag number of the address to be accessed.

Varying Associativity Over Cache Size

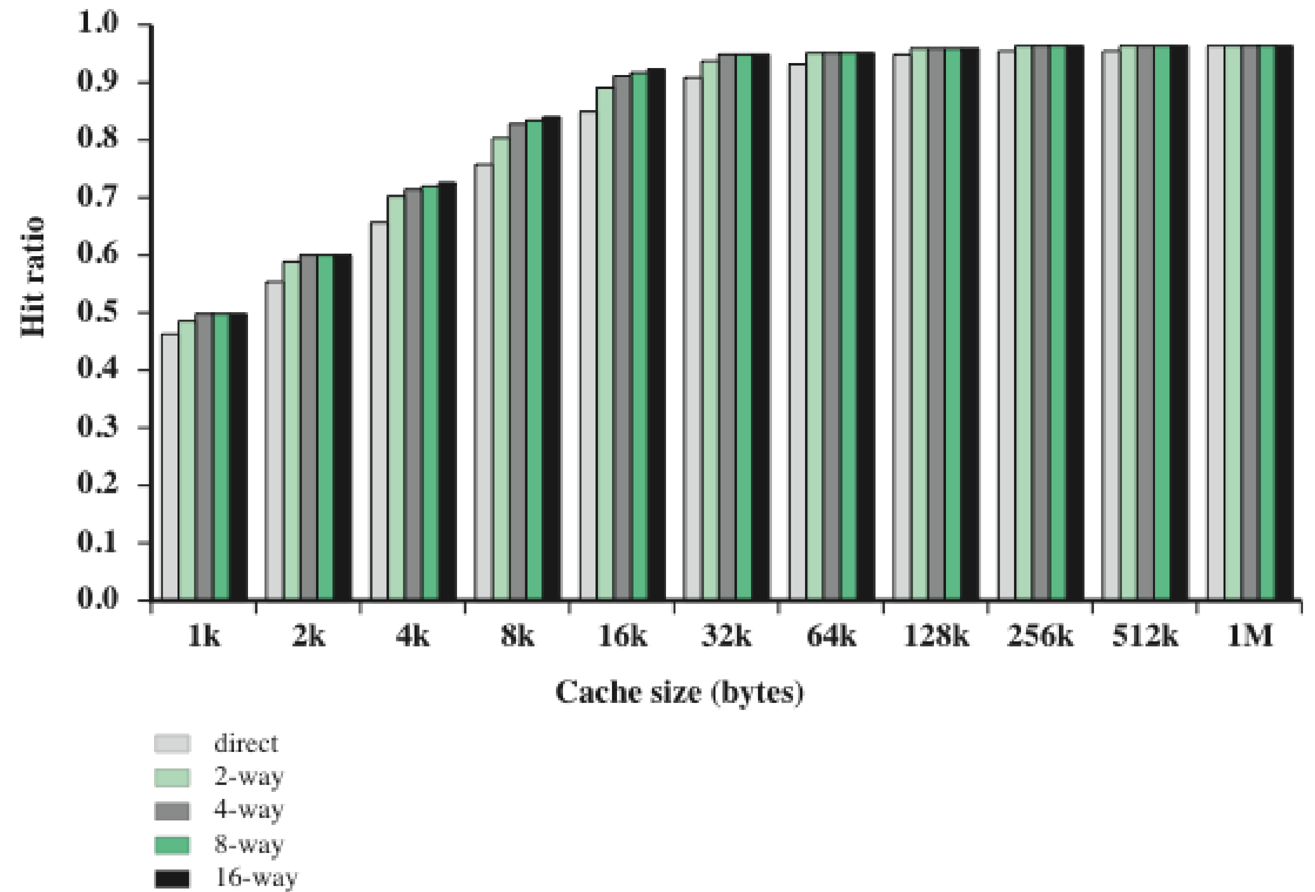


Figure 4.16 Varying Associativity over Cache Size

Replacement Algorithms

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
- For direct mapping there is only one possible line for any particular block and no choice is possible
- For the associative and set-associative techniques a replacement algorithm is needed
- To achieve high speed, an algorithm must be implemented in hardware

AAiT

The four most common replacement algorithms are:

- Least recently used (LRU)
  - Most effective
  - Replace that block in the set that has been in the cache longest with no reference to it
  - Because of its simplicity of implementation, LRU is the most popular replacement algorithm
- First-in-first-out (FIFO)
  - Replace that block in the set that has been in the cache longest
  - Easily implemented as a round-robin or circular buffer technique
- Least frequently used (LFU)
  - Replace that block in the set that has experienced the fewest references
  - Could be implemented by associating a counter with each line
- Random
  - A technique not based on usage, it picks a line at random from among the candidate lines
  - Simulation studies have shown that random replacement provides only slightly inferior performance to an algorithm based on usage

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 (when an I/O module uses DMA and modify RAM ➔ cache word is invalid and the vice versa)

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

Write Through and Write Back

- Write through
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
  - The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck
- Write back
  - Minimizes memory writes
  - Updates are made only in the cache
  - Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - This makes for complex circuitry and a potential bottleneck

If more than one device (typically a processor) has a separate cache but share main memory, a new problem is introduced. If data in one cache are altered, even if a write-through policy is used, the other caches may contain invalid data. A system that prevents this problem is said to maintain cache coherency:

- **Bus watching with write through:** Each cache controller monitors the address lines to detect write operations to memory by other bus masters. If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry.
- **Hardware transparency:** Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches.
- **Non-cacheable memory:** Only a portion of main memory is shared by more than one processor, and this is designated as non-cacheable.

AAiT

Line Size

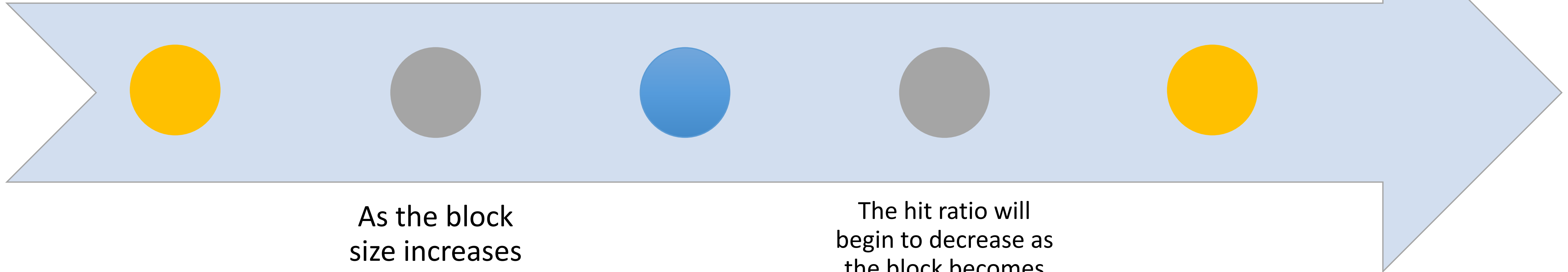
AAiT

When a block of data is retrieved and placed in the cache not only the desired word but also some number of adjacent words are retrieved

As the block size increases more useful data are brought into the cache

Two specific effects come into play:

- Larger blocks reduce the number of blocks that fit into a cache
- As a block becomes larger each additional word is farther from the requested word



As the block size increases the hit ratio will at first increase because of the principle of locality

The hit ratio will begin to decrease as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced

A size of from 8 to 64 bytes seems reasonably close to optimum. For HPC systems, 64- and 128-byte cache line sizes are most frequently used.

Multilevel Caches

- As logic density has increased it has become possible to have a cache on the same chip as the processor
- The on-chip cache reduces the processor's external bus activity and speeds up execution time and increases overall system performance
  - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
  - On-chip cache accesses will complete appreciably faster than would even zero-wait state bus cycles
  - During this period the bus is free to support other transfers
- Two-level cache:
  - Internal cache designated as level 1 (L1)
  - External cache designated as level 2 (L2)
- Potential savings due to the use of an L2 cache depends on the hit rates in both the L1 and L2 caches
- The use of multilevel caches complicates all of the design issues related to caches, including size, replacement algorithm, and write policy

Hit Ratio (L1 & L2) For 8 Kbyte and 16 Kbyte L1

- The need for the L2 cache to be larger than the L1 cache to affect performance makes sense. If the L2 cache has the same line size and capacity as the L1 cache, its contents will more or less mirror those of the L1 cache.

- With the increasing availability of on-chip area, most contemporary microprocessors have moved the L2 cache onto the processor chip and added an L3 cache accessible over the external bus.
- More recently, most microprocessors have incorporated an on-chip L3 cache. In either case, there appears to be a performance advantage to adding the third level. Further, large systems now incorporate 3 on-chip cache levels and a fourth level of cache shared across multiple chips

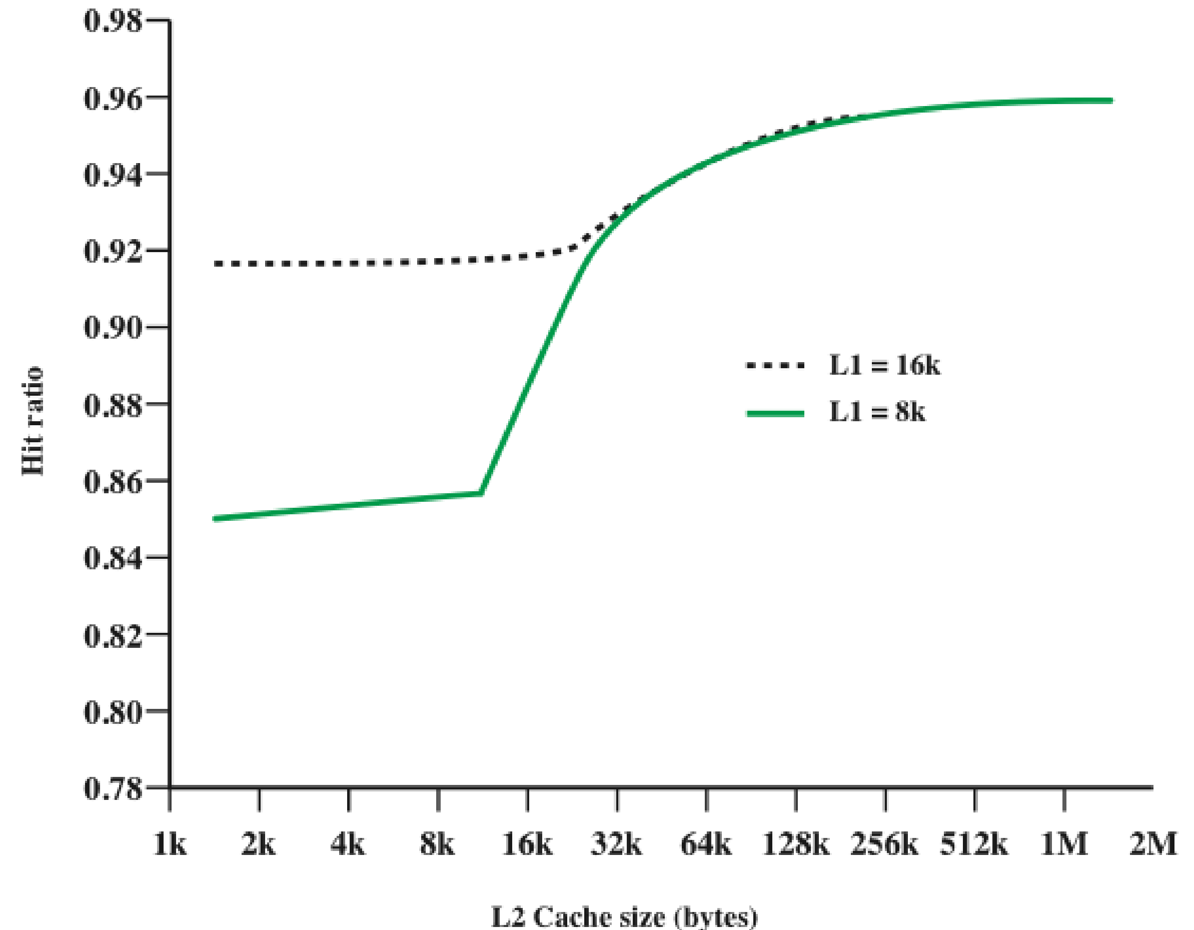


Figure 4.17 Total Hit Ratio (L1 and L2) for 8 Kbyte and 16 Kbyte L1

AAiT

Unified Versus Split Caches

- Has become common to split cache:
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
- Advantages of unified cache:
  - Higher hit rate
    - Balances load of instruction and data fetches automatically
    - Only one cache needs to be designed and implemented
- Trend is toward split caches at the L1 and unified caches for higher levels
- Advantages of split cache:
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining

AAiT

Pentium 4 Cache

Problem	Solution	Processor on Which Feature First Appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip.	Add external L2 cache using faster technology than main memory.	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

Table 4.4 Intel Cache Evolution

Pentium 4 Block Diagram

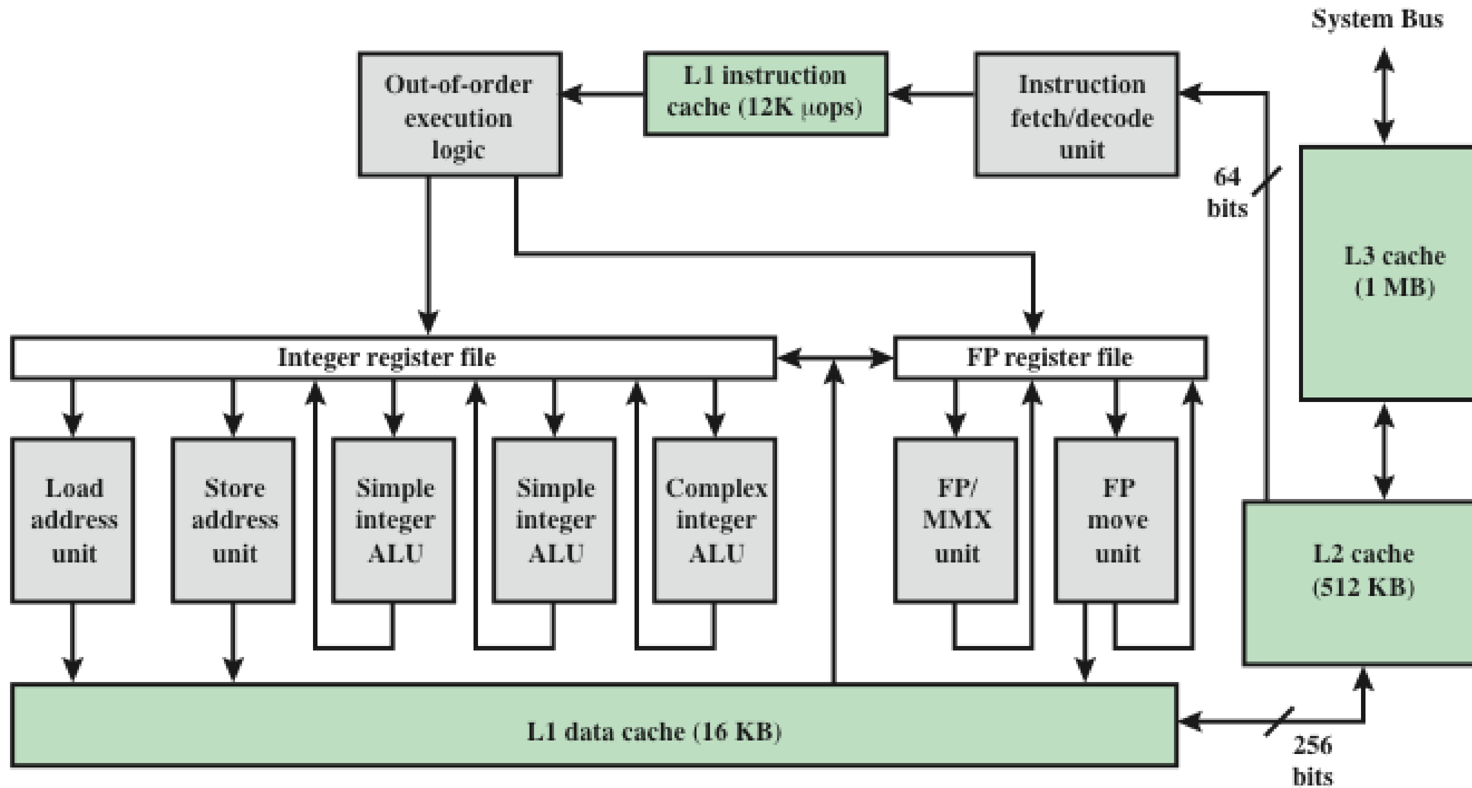


Figure 4.18 Pentium 4 Block Diagram

- **Fetch/decode unit:** Fetches program instructions from L2, decodes and stores the results in the L1 instruction cache.
- **Out-of-order execution logic:** Schedules execution of the decoded micro-operations subject to data dependencies and resource availability. As time permits, this unit schedules speculative execution of micro-operations that may be required in the future.
- **Execution units:** These units executes micro-operations, fetching the required data from the L1 data cache and temporarily storing results in registers.
- **Memory subsystem:** This unit includes the L2 and L3 caches and the system bus, which is used to access main memory when the L1 and L2 caches have a cache miss and to access the system I/O resources.

Pentium 4 Cache Operating Modes

- The L1 data cache is controlled by two bits in one of the control registers, labeled the CD (cache disable) and NW (not write-through) bits.
- There are two Pentium 4 instructions that can be used to control the data cache:
  - INVD: invalidates (flushes) the internal cache memory and signals the external cache to invalidate.
  - WBINVD writes back and invalidates internal cache and then writes back and invalidates external cache.
- Both the L2 and L3 caches are eight-way set-associative with a line size of 128 bytes.

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.

Table 4.5 Pentium 4 Cache Operating Modes

ARM Cache Features

Core	Cache Type	Cache Size (kB)	Cache Line Size (words)	Associativity	Location	Write Buffer Size (words)
ARM720T	Unified	8	4	4-way	Logical	8
ARM920T	Split	16/16 D/I	8	64-way	Logical	16
ARM926EJ-S	Split	4-128/4-128 D/I	8	4-way	Logical	16
ARM1022E	Split	16/16 D/I	8	64-way	Logical	16
ARM1026EJ-S	Split	4-128/4-128 D/I	8	4-way	Logical	8
Intel StrongARM	Split	16/16 D/I	4	32-way	Logical	32
Intel Xscale	Split	32/32 D/I	8	32-way	Logical	32
ARM1136-JF-S	Split	4-64/4-64 D/I	8	4-way	Physical	32

Table 4.6 ARM Cache Features

AAiT

ARM Cache and Write Buffer Organization

- An interesting feature of the ARM architecture is the use of a small first-in-first-out (FIFO) write buffer to enhance memory write performance.
- When the processor performs a write to a bufferable area, the data are placed in the write buffer at processor clock speed and the processor continues execution.
- A write occurs when data in the cache are written back to main memory

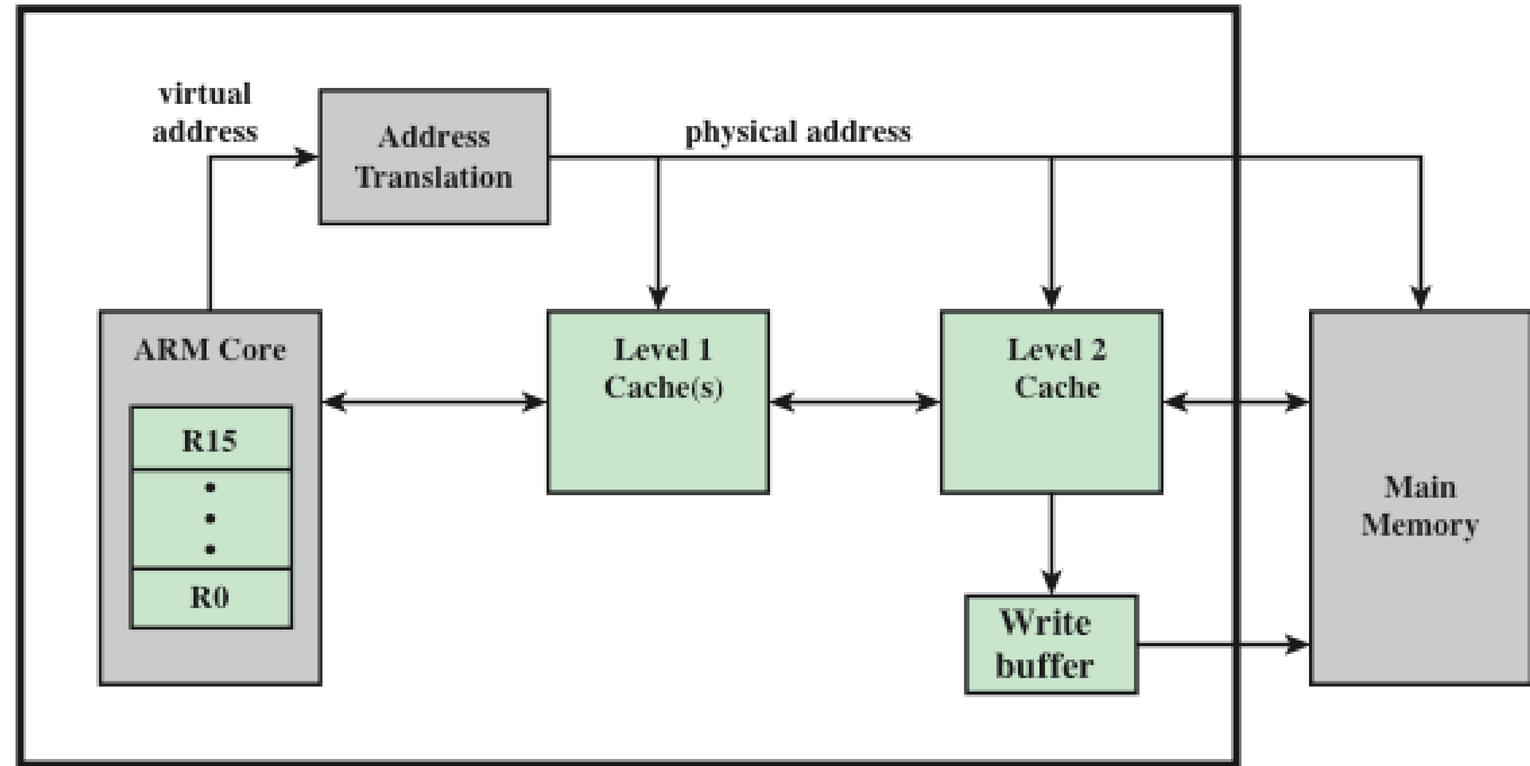


Figure 4.19 ARM Cache and Write Buffer Organization

AAiT

Summary

Chapter 4 Cache Memory

- Characteristics of Memory Systems
  - Location
  - Capacity
  - Unit of transfer
- Memory Hierarchy
  - How much?
  - How fast?
  - How expensive?
- Cache memory principles
- Elements of cache design
  - Cache addresses
  - Cache size
  - Mapping function
  - Replacement algorithms
  - Write policy
  - Line size
  - Number of caches
- Pentium 4 cache organization
- ARM cache organization

William Stallings
Computer Organization and Architecture
9th Edition