The Memory System

Overview

- Basic memory circuits
- Organization of the main memory
- Cache memory concept
- Virtual memory mechanism
- Secondary storage

Some Basic Concepts

Traditional Architecture

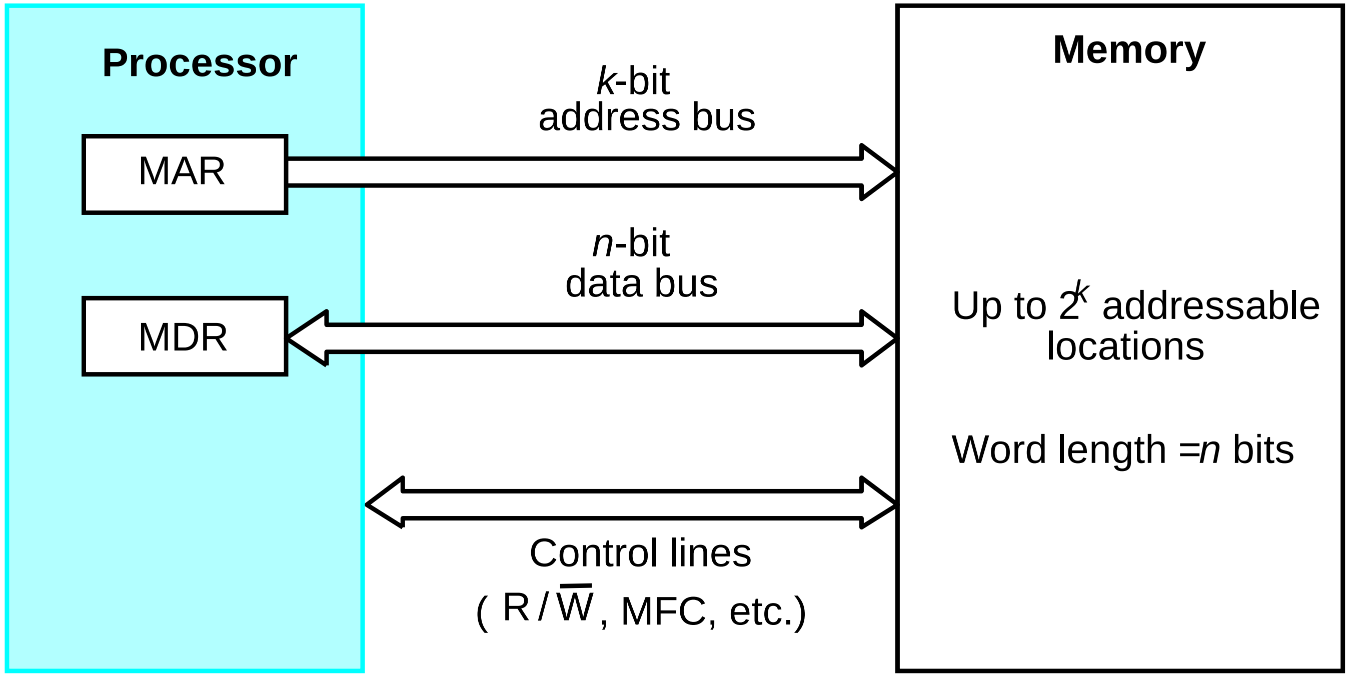


Figure 5.1. Connection of the memory to the processor.

Basic Concepts

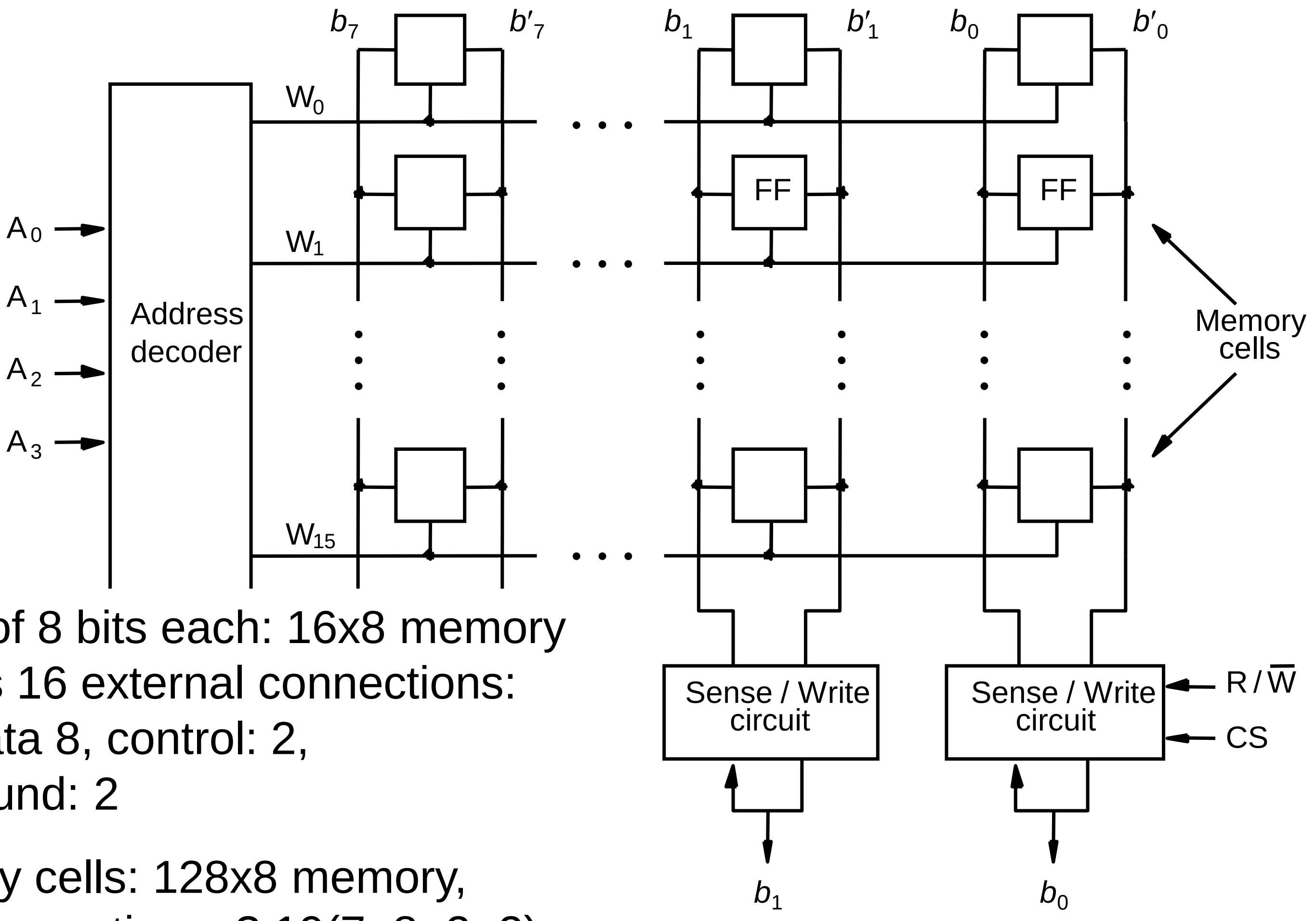
- "Block transfer" – bulk data transfer
- *Memory access time*-time that elapses between the initiation of an operation to transfer a word of data and the completion of that operation.
- *Memory cycle time*- the minimum time delay required between the initiation of two successive memory operations
- RAM – any location can be accessed for a Read or Write operation in some fixed amount of time that is independent of the location's address.

Memory

- **Cache memory**-a small, fast memory inserted between the larger, slower main memory and the processor.
  - It holds the currently active portions of a program and their data.
- **Virtual memory** -only the active portions of a program are stored in the main memory, and the remainder is stored on the much larger secondary storage device.
- Sections of the program are transferred back and forth between the main memory and the secondary storage device in a manner that is transparent to the application program.
- the application program sees a memory that is much larger than the computer’s physical main memory

Semiconductor RAM Memories

Internal Organization of Memory Chips



16 words of 8 bits each: 16x8 memory org.. It has 16 external connections: addr. 4, data 8, control: 2, power/ground: 2

1K memory cells: 128x8 memory, external connections: ? 19(7+8+2+2)

1Kx1:? 15 (10+1+2+2)

ization of bit cells in a memory chip.

A Memory Chip

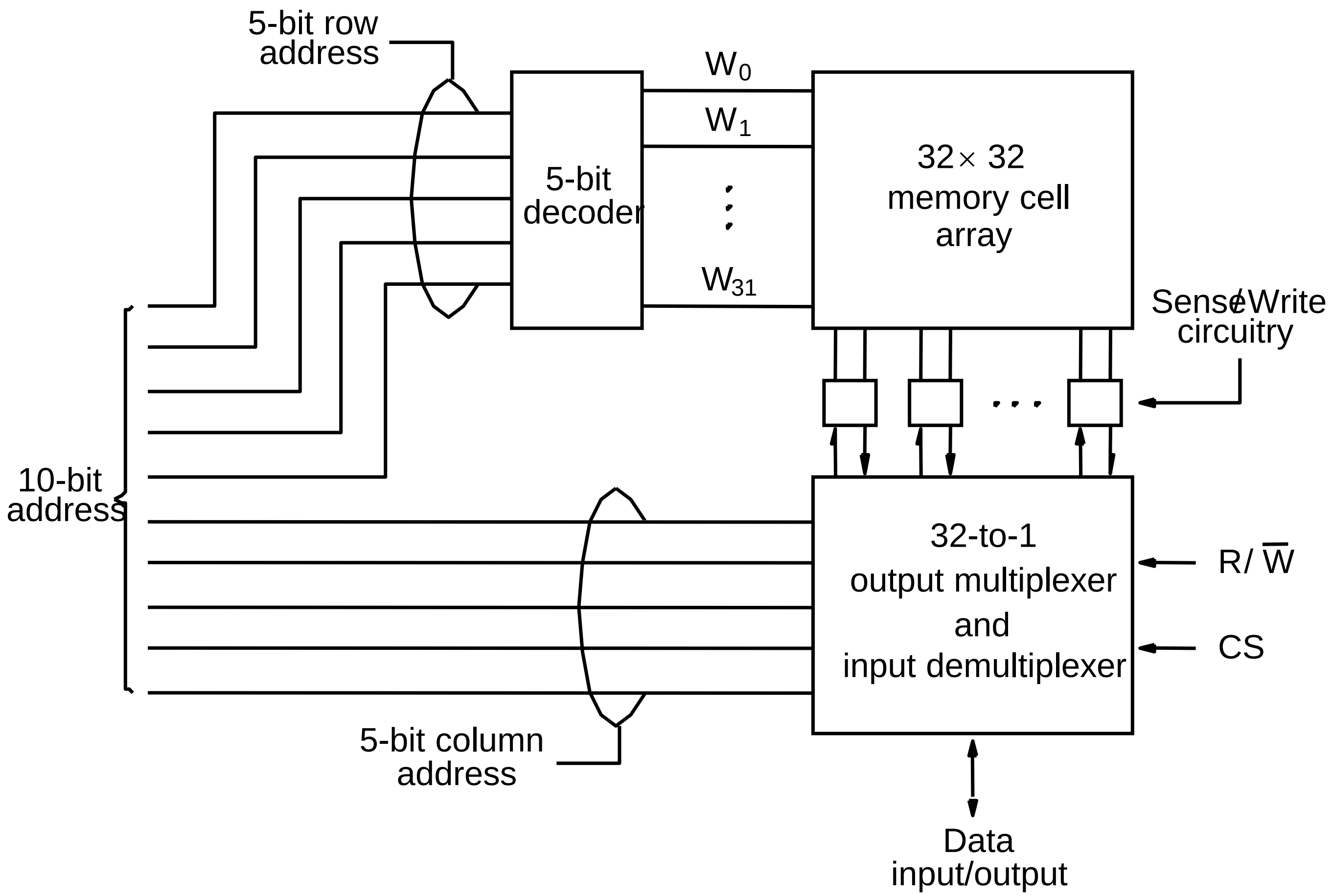


Figure 5.3. Organization of a 1K × 1 memory chip.

Structure of Larger Memoriies

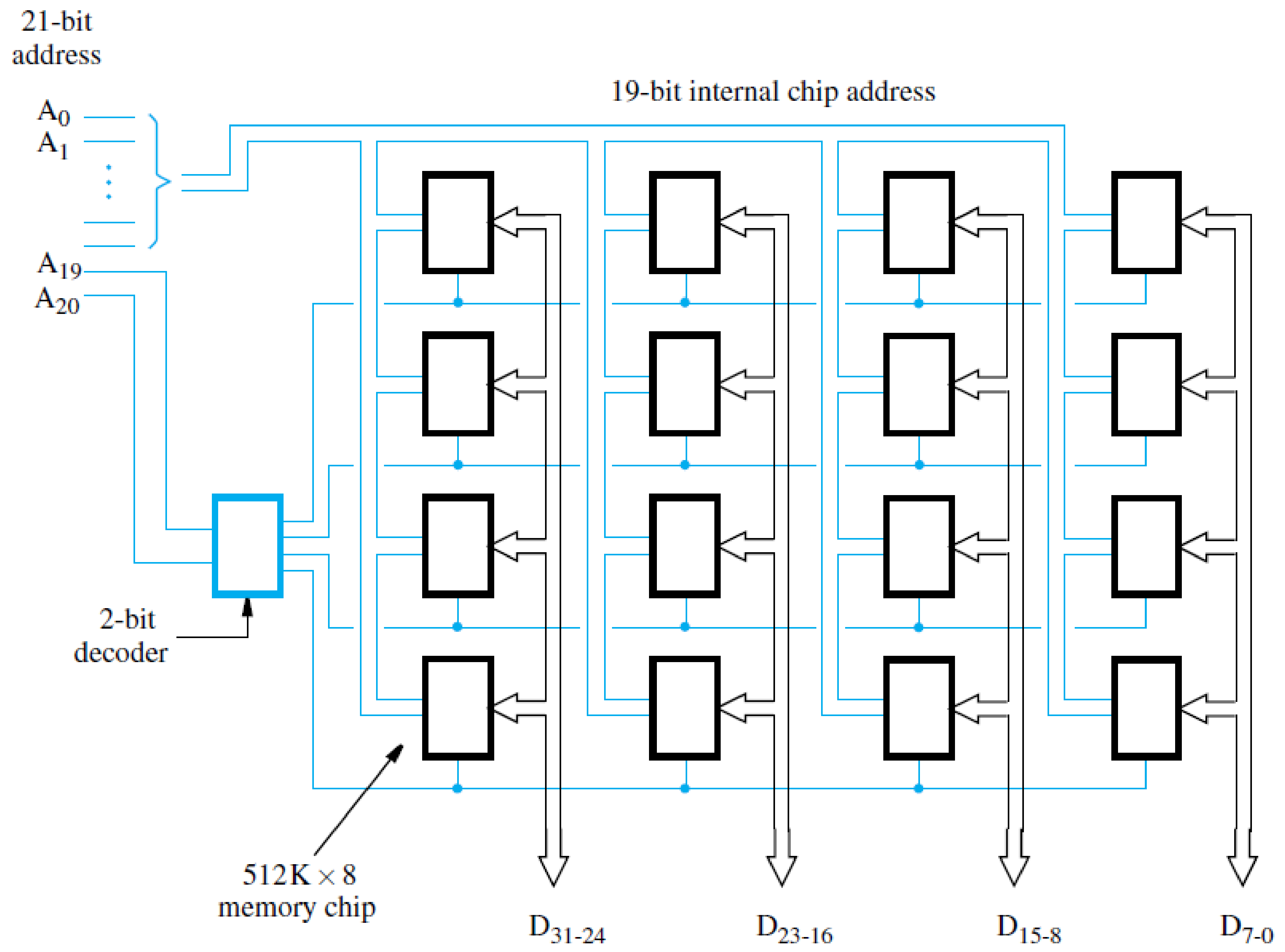
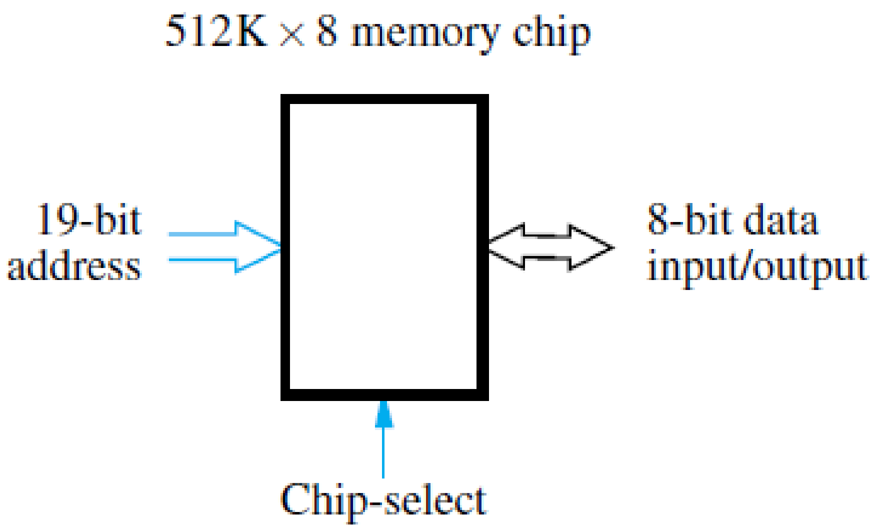


Figure 8.10 Organization of a 2M × 32 memory module using 512K × 8 static memory chips.


512K × 8 memory chip
19-bit address
8-bit data input/output
Chip-select

Memory Hierarchy Speed, Size, and Cost

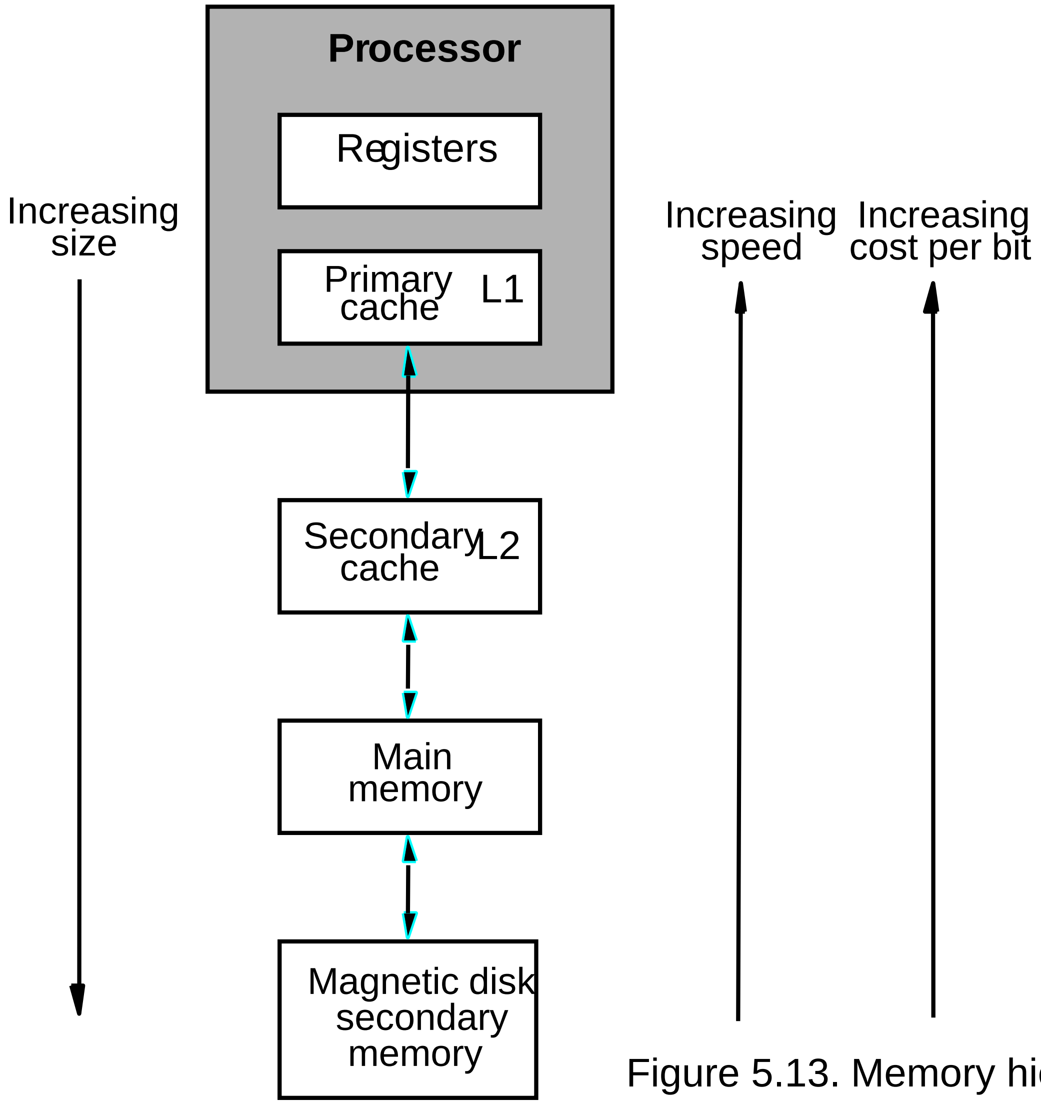


Figure 5.13. Memory hierarchy.

Cache Memories

Cache

- What is cache?
- Why we need it?
- Locality of reference (very important)

  - temporal- a recently executed instruction is likely to be executed again very soon. that.

  - spatial- instructions close to a recently executed instruction are also likely to be executed soon
- Cache block – *cache line*
  - *A set of contiguous address locations of some size*

Cache

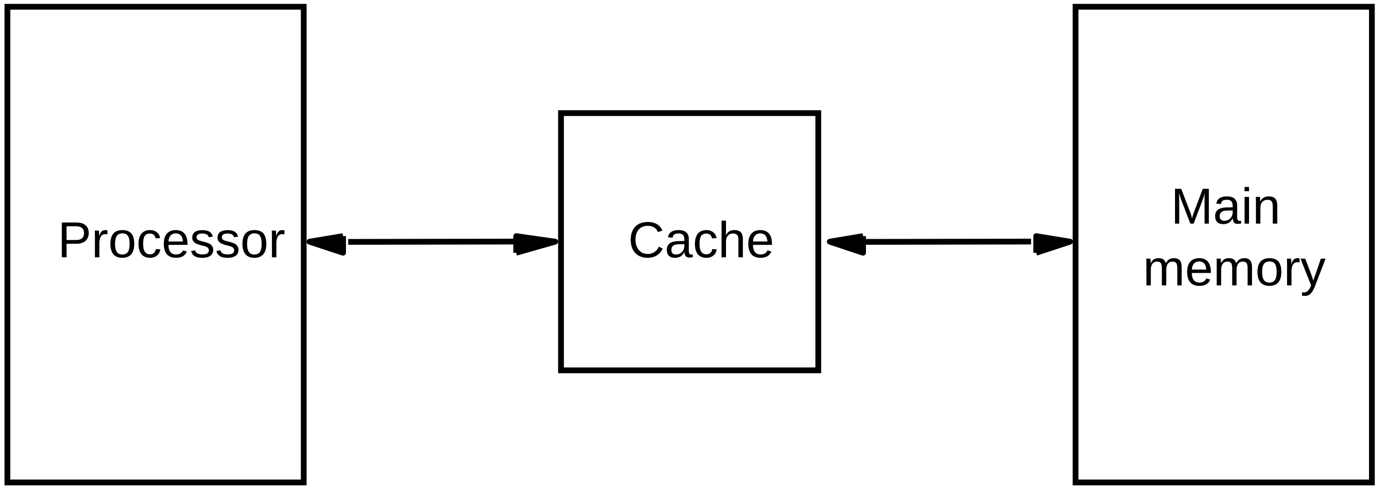


Figure 5.14. Use of a cache memory.

- Replacement algorithm
- Hit / miss
- Write-through / Write-back
- Load through

- The correspondence between the main memory blocks and those in the cache is specified by a *mapping function*.
- When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constitutes the cache's *replacement algorithm*.

Cache Hits

- The cache control circuitry determines whether the requested word currently exists in the cache.
- If it does, the Read or Write operation is performed on the appropriate cache location. -a ***read* or *write hit***.
  - main memory not involved when there is a cache hit in a Read operation
  - Write Operation –two ways
  - ***write-through* protocol**, both the cache location and the main memory location are updated.
  - ***write-back*, or *copy-back*,** protocol-update only the cache location and to mark the block containing it with an associated flag bit, often called the ***dirty* or *modified*** *bit*. The main memory location of the word is updated later

Comparison

- The write-through protocol is simpler than the write-back protocol- but it results in unnecessary Write operations in the main memory when a given cache word is updated several times during its cache residency.
- The write-back protocol also involves unnecessary Write operations, because all words of the block are eventually written back, even if only a single word has been changed while the block was in the cache.
- The write-back protocol is used most often, to take advantage of the high speed with which data blocks can be transferred to memory chips.

Cache Misses

- Read operation for a word that is not in the cache constitutes a *Read miss*.
- causes the block of words containing the requested word to be copied from the main memory into the cache.
- After the entire block is loaded into the cache, the particular word requested is forwarded to the processor.
- Alternatively, this word may be sent to the processor as soon as it is read from the main memory. The latter approach, which is called ***load-through*, or *early restart*,** reduces the processor’s waiting time somewhat, at the expense of more complex circuitry.

- When a *Write miss* occurs in a computer that uses the write-through protocol, the information is written directly into the main memory.
- For the write-back protocol, the block containing the addressed word is first brought into the cache, and then the desired word in the cache is overwritten with the new information.

Direct Mapping

Block j of main memory maps onto block j modulo 128 of the cache

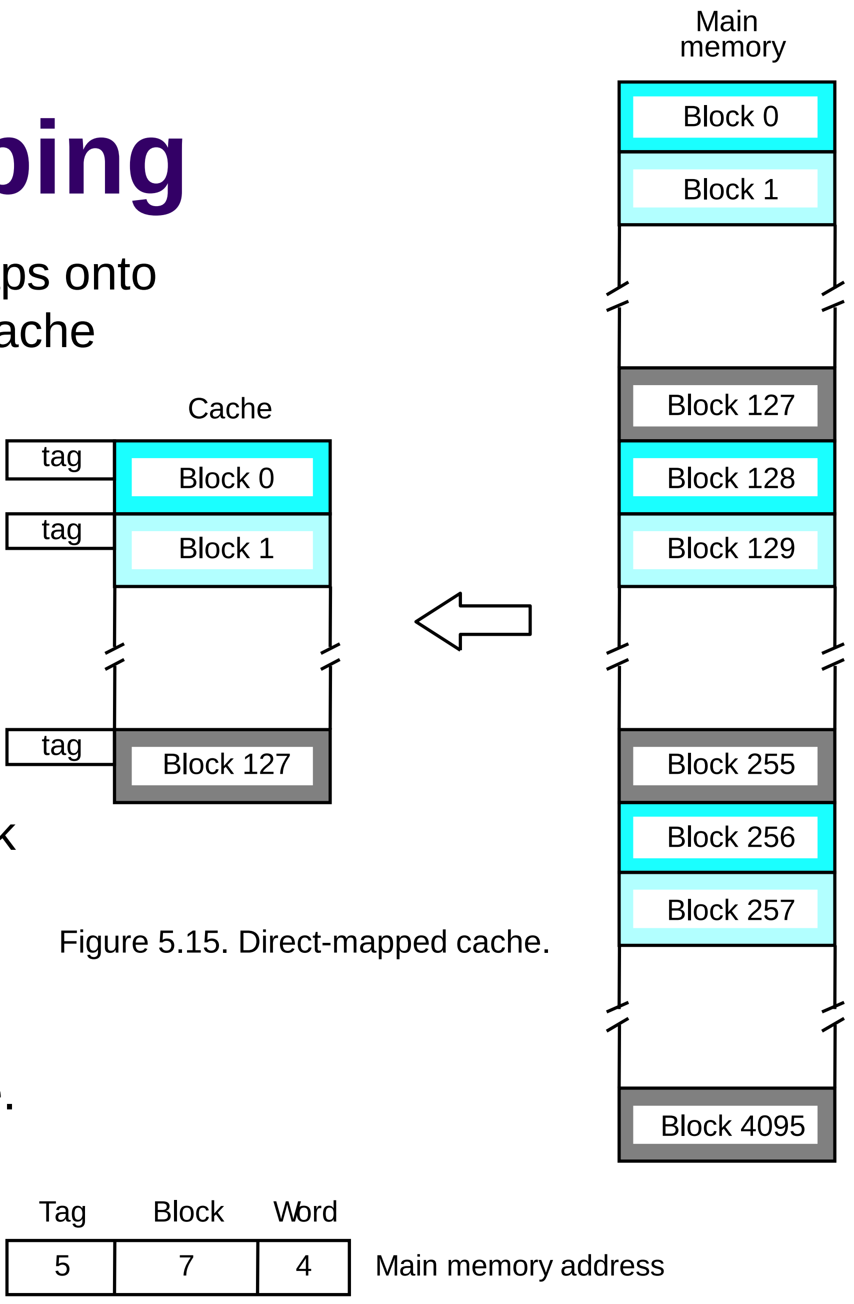


Figure 5.15. Direct-mapped cache.

4: one of 16 words. (each block has $16=2^4$ words)

7: points to a particular block in the cache ($128=2^7$)

5: 5 tag bits are compared with the tag bits associated with its location in the cache. Identify which of the 32 blocks that are resident in the cache (4096/128).

Direct Mapping

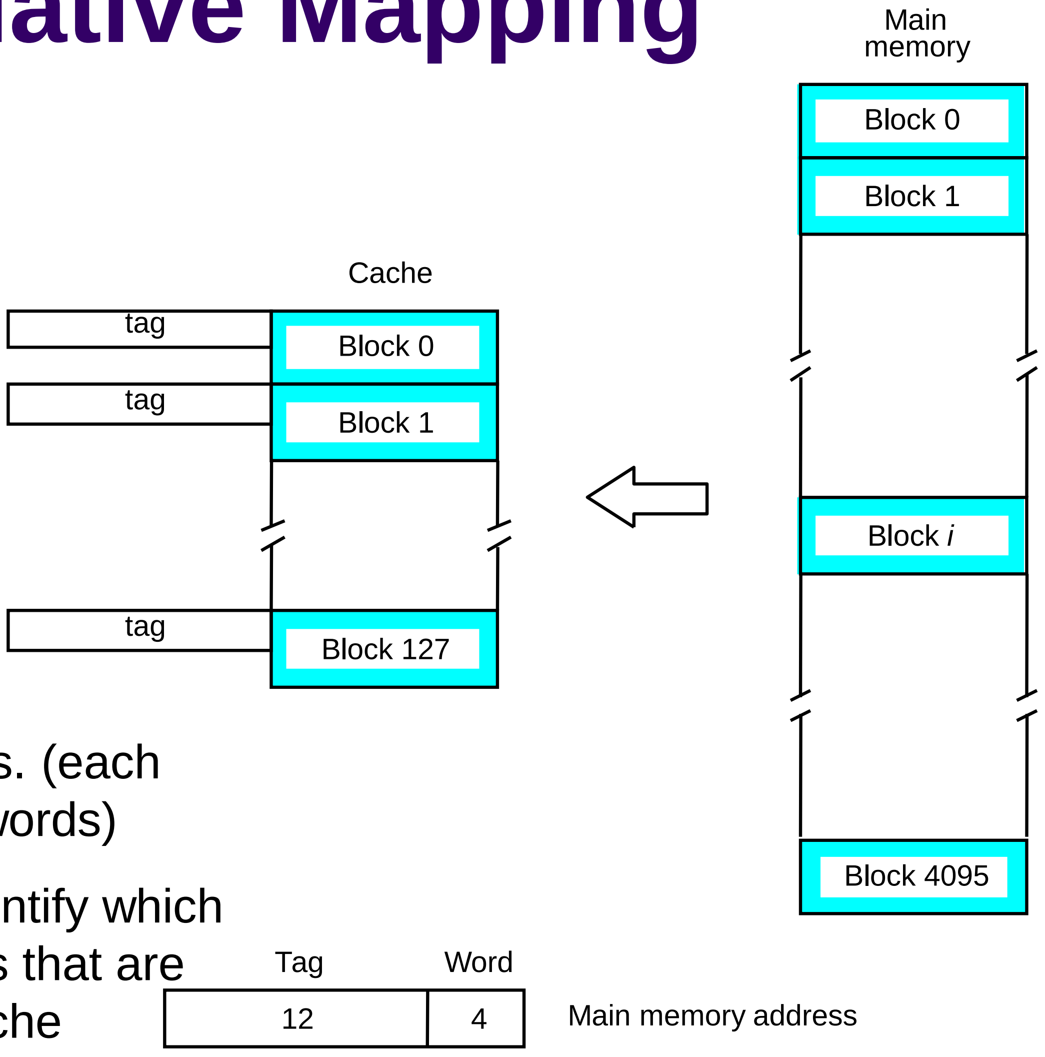
Tag	Block	Word
5	7	4

Main memory address

11101,1111111,1100

- Tag: 11101
- Block: 1111111=127, in the 127th block of the cache
- Word:1100=12, the 12th word of the 127th block in the cache

Associative Mapping

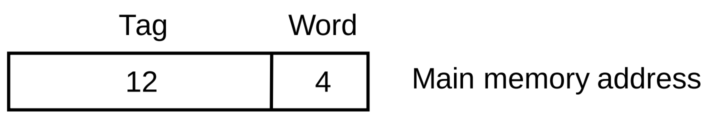


4: one of 16 words. (each block has $16=2^4$ words)

12: 12 tag bits Identify which of the 4096 blocks that are resident in the cache $4096=2^{12}$.

Figure 5.16. Associative-mapped cache.

Associative Mapping



111011111111,1100

- Tag: 111011111111
- Word:1100=12, the 12th word of a block in the cache

Set-Associative Mapping

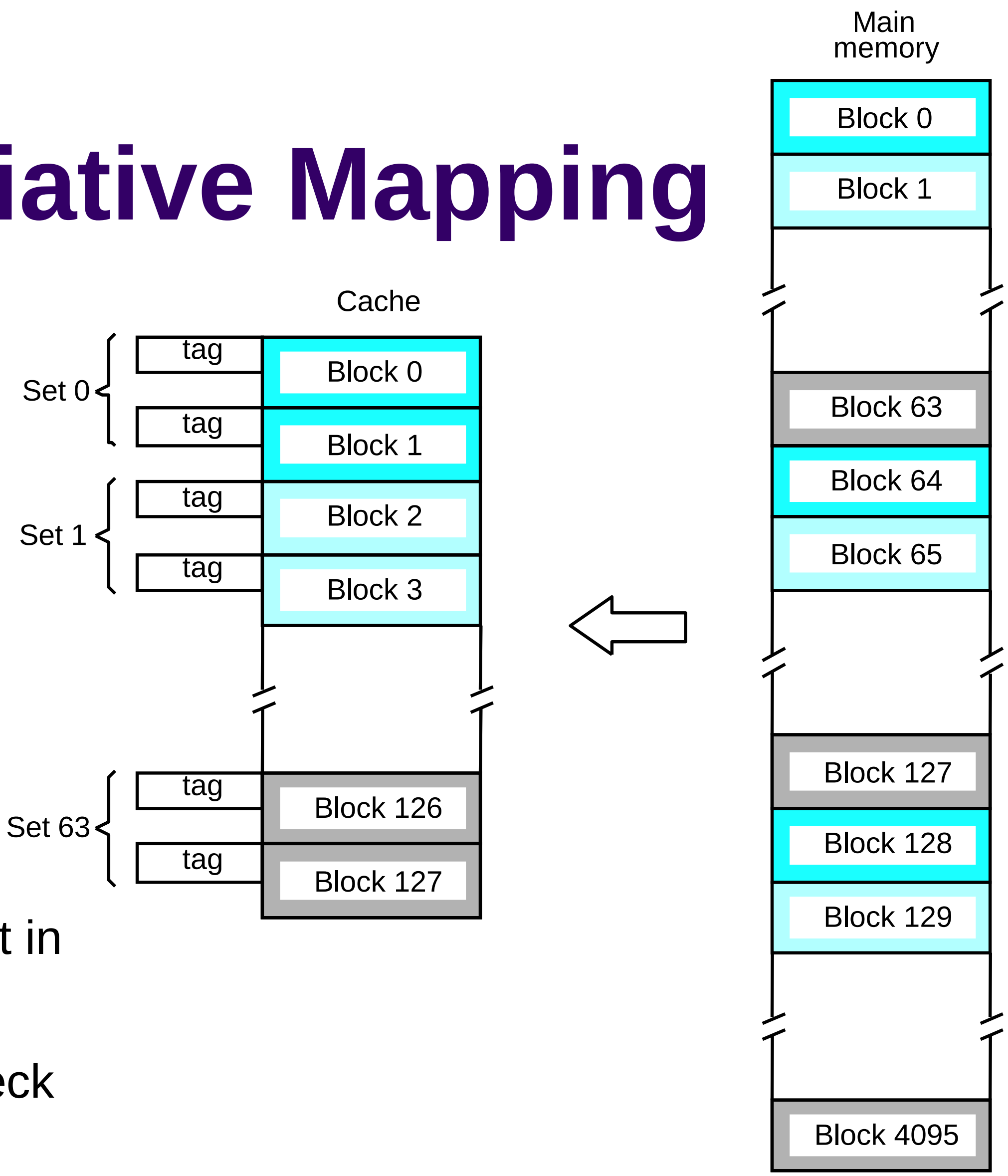


Figure 5.17. Set-associative-mapped cache with two blocks per set.

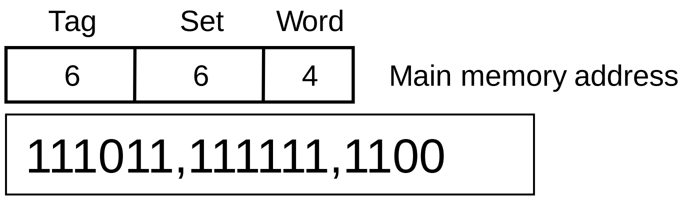
Tag	Set	Word	
6	6	4	Main memory address

4: one of 16 words. (each block has $16=2^4$ words)

6: points to a particular set in the cache ($128/2=64=2^6$)

6: 6 tag bits is used to check if the desired block is present ($4096/64=2^6$).

Set-Associative Mapping



- Tag: 111011
- Set: 111111=63, in the 63th set of the cache
- Word:1100=12, the 12th word of the 63th set in the cache

- A block-set-associative cache consists of a total of 64 blocks, divided into 4-block sets.The main memory contains 4096 blocks, each consisting of 32 words. Assuming a 32-bit byte-addressable address space, how many bits are there in each of the Tag, Set, and Word fields?

Solution:

Number of sets = 64/4 = 16

Set bits = 4(2^4 = 16)

Number of bytes = 128(Assuming 4 byte word)

Word bits = 7 bits (2^7 = 128)

- Tag=32-(4+7)=21

Problem:

A block-set-associative cache consists of a total of 64 blocks, divided into 4-block sets. The main memory contains 4096 blocks, each consisting of 128 words.

a)How many bits are there in MM address?

b)How many bits are there in each of the TAG, SET & word fields

Solution:

Number of sets = 64/4 = 16

Set bits = 4(2^4 = 16)

Number of words = 128

Word bits = 7 bits (2^7 = 128)

MM capacity : 4096 x 128 ($2^{12} \times 2^7 = 2^{19}$)

a)Number of bits in memory address = 19 bits

b)

8	4	7

TAG SET WORD

TAG bits = 19 – (7+4) = 8 bits.

Problem:

A computer system has a MM capacity of a total of 1M 16 bits words. It also has

a 4K words cache organized in the block set associative manner, with 4 blocks per

set & 64 words per block. Calculate the number of bits in each of the TAG, SET & WORD fields of MM address format

Solution:

Capacity: 1M (220 = 1M)

Number of words per block = 64

Number of blocks in cache = 4k/64 = 64

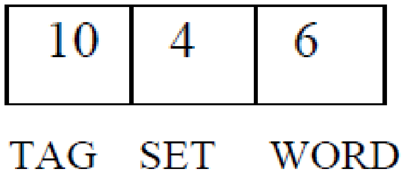
Number of sets = 64/4 = 16

Set bits = 4 (2^4 = 16)

Word bits = 6 bits (2^6 = 64)

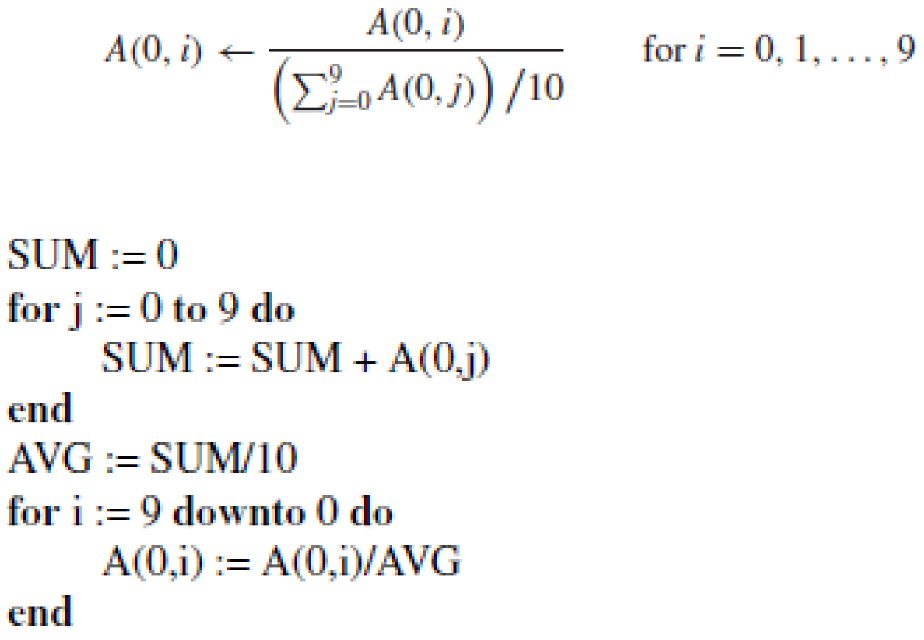
Tag bits = 20-(6+4) = 10 bits

MM address format



10	4	6
TAG	SET	WORD

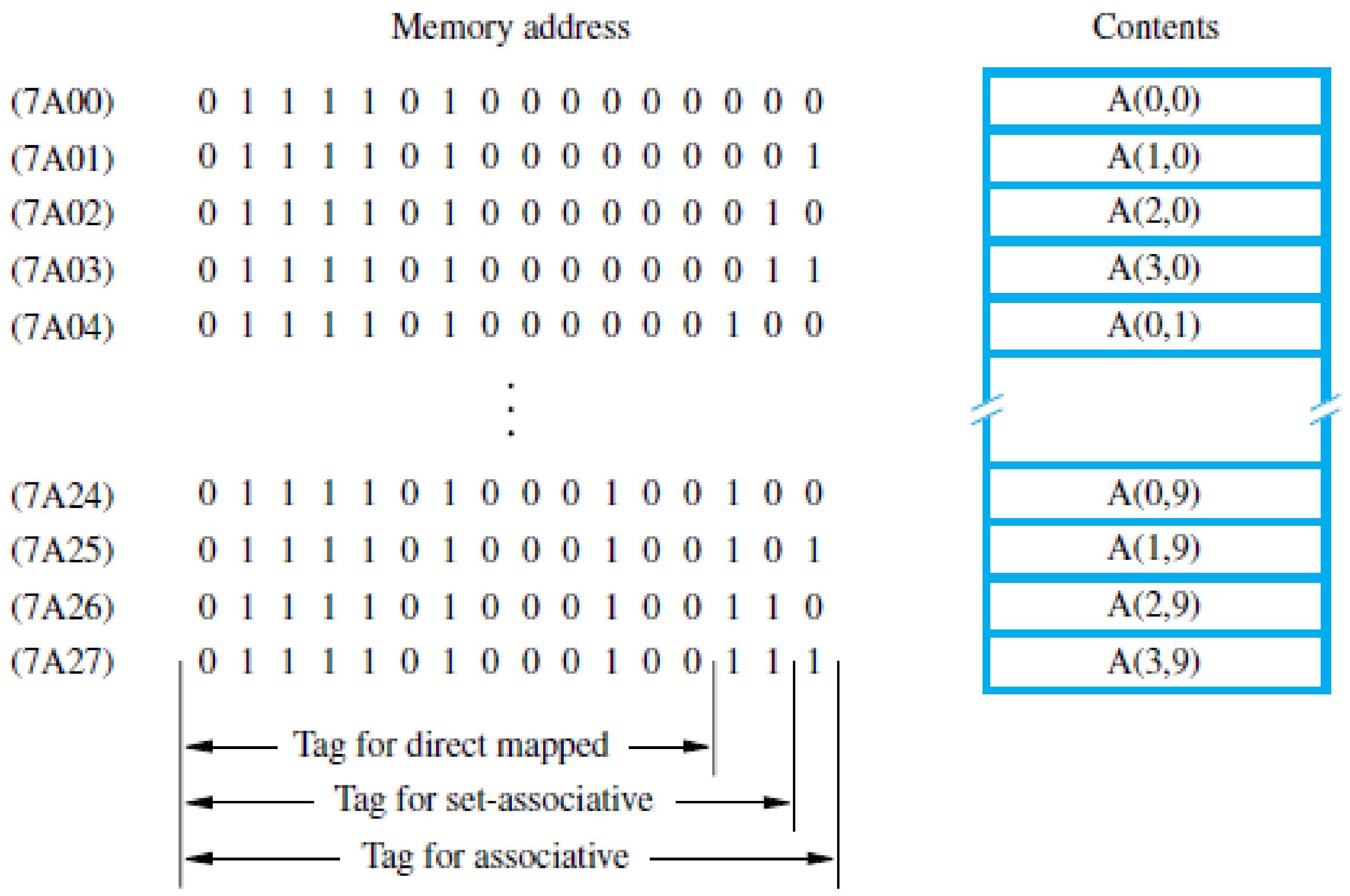
- A 4 × 10 array of numbers, each occupying one word, is stored in main memory locations 7A00 through 7A27 (hex). The elements of this array, A, are stored in column order, as shown below. Assume the data cache has space for only eight blocks of data.



$$A(0,i) \leftarrow \frac{A(0,i)}{\left(\sum_{j=0}^{9} A(0,j)\right)/10} \qquad \text{for } i = 0, 1, \ldots, 9$$

```
SUM := 0
for j := 0 to 9 do
        SUM := SUM + A(0,j)
end
AVG := SUM/10
for i := 9 downto 0 do
        A(0,i) := A(0,i)/AVG
end
```

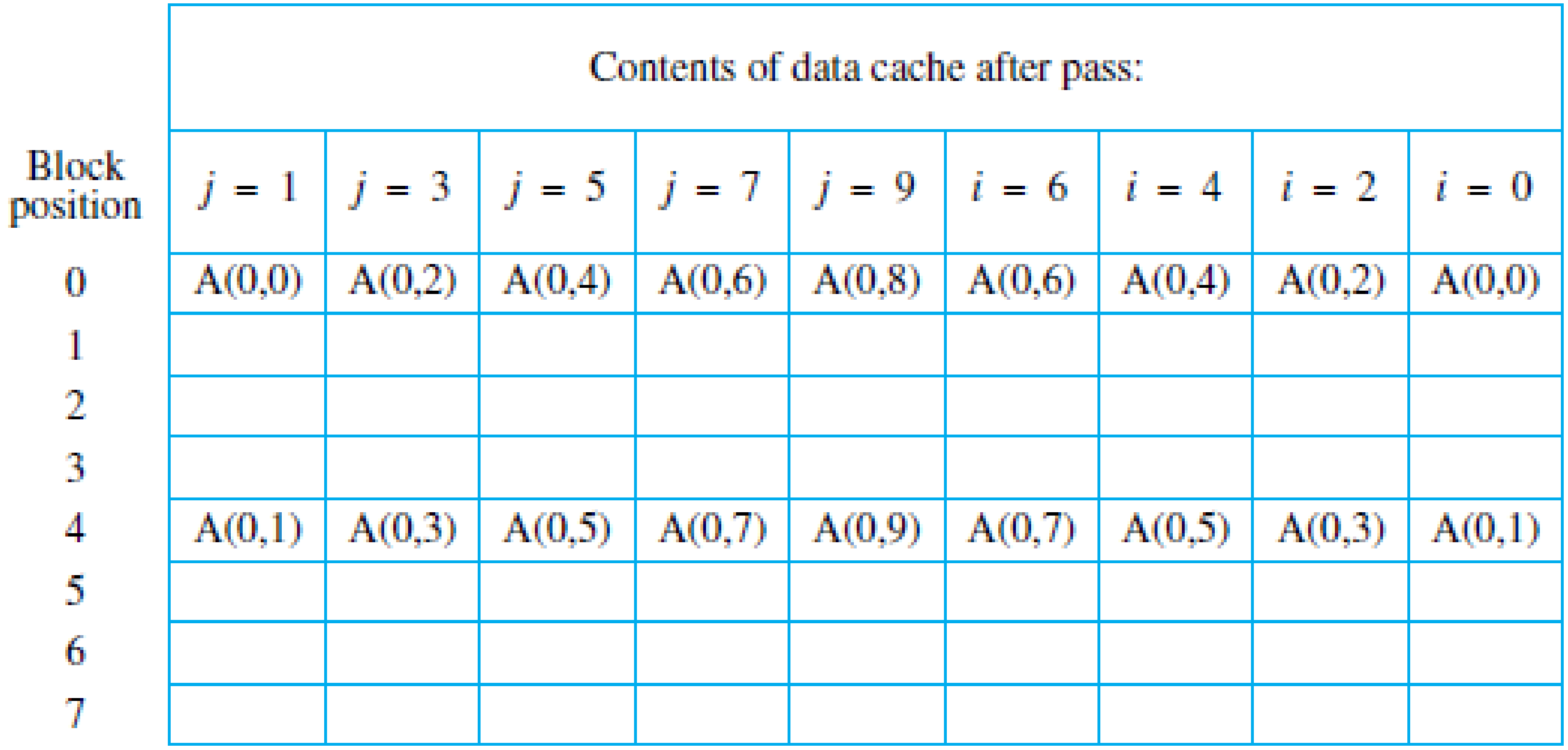
Figure 8.20 Task for example in Section 8.6.3.



	Memory address	Contents
(7A00)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	A(0,0)
(7A01)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1	A(1,0)
(7A02)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0	A(2,0)
(7A03)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 1	A(3,0)
(7A04)	0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0	A(0,1)
	⋮	
(7A24)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0	A(0,9)
(7A25)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 1	A(1,9)
(7A26)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 0	A(2,9)
(7A27)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 1	A(3,9)

Figure 8.19 An array stored in the main memory.

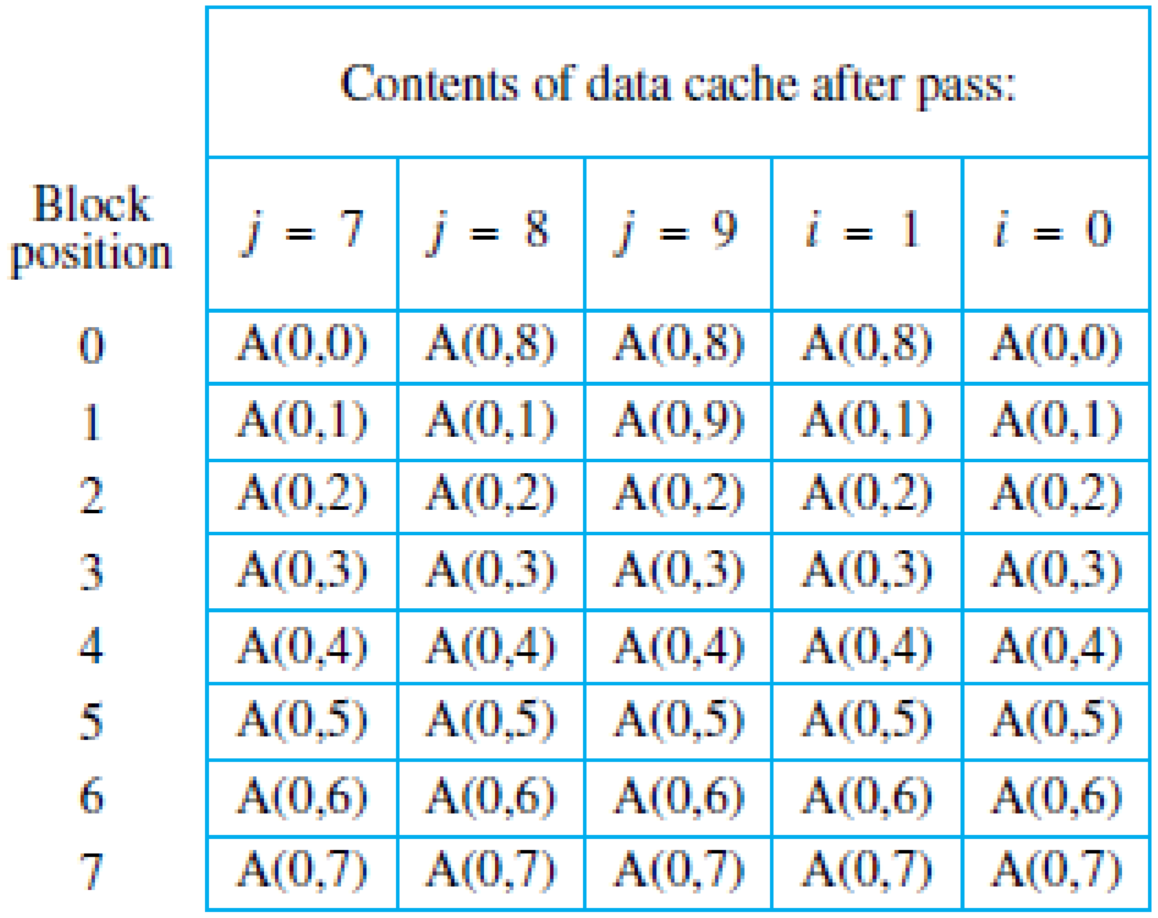
Direct Mapped Cache



	Contents of data cache after pass:								
Block position	$j = 1$	$j = 3$	$j = 5$	$j = 7$	$j = 9$	$i = 6$	$i = 4$	$i = 2$	$i = 0$
0	A(0,0)	A(0,2)	A(0,4)	A(0,6)	A(0,8)	A(0,6)	A(0,4)	A(0,2)	A(0,0)
1									
2									
3									
4	A(0,1)	A(0,3)	A(0,5)	A(0,7)	A(0,9)	A(0,7)	A(0,5)	A(0,3)	A(0,1)
5									
6									
7									

Figure 8.21 Contents of a direct-mapped data cache.

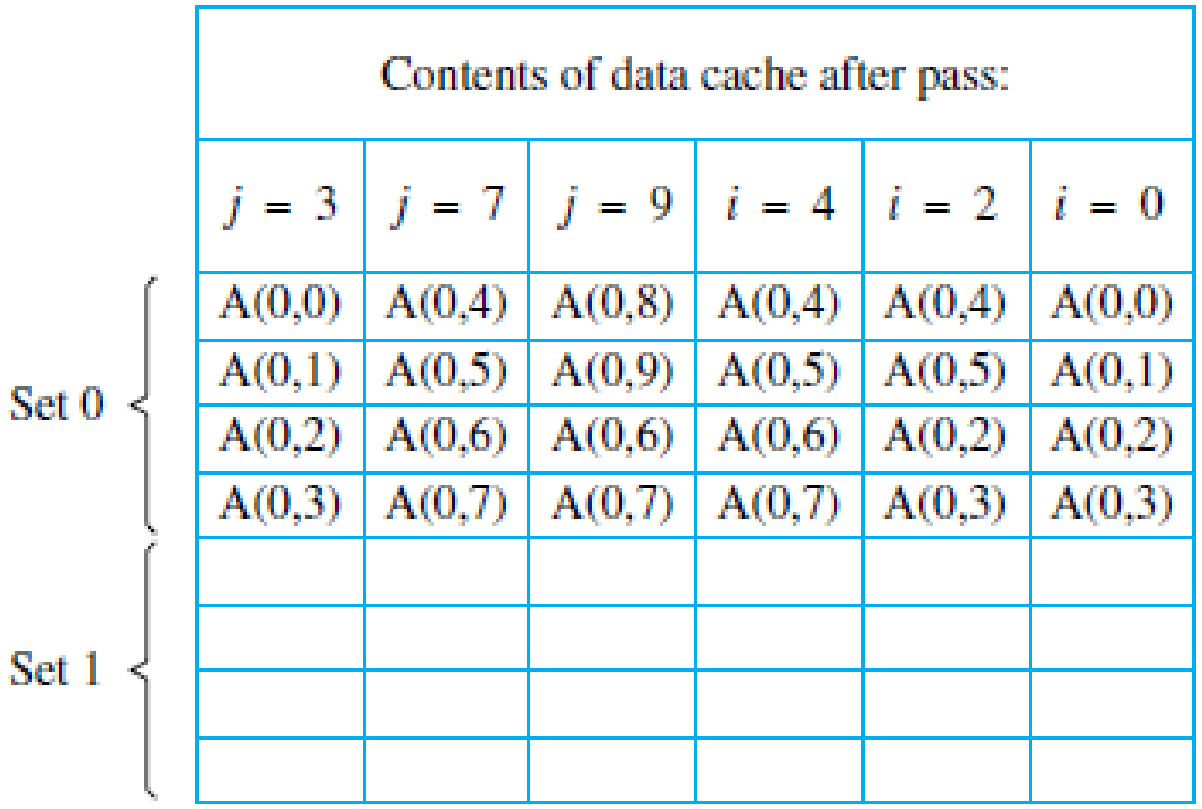
Associative Mapped Cache



	Contents of data cache after pass:				
Block position	$j = 7$	$j = 8$	$j = 9$	$i = 1$	$i = 0$
0	A(0,0)	A(0,8)	A(0,8)	A(0,8)	A(0,0)
1	A(0,1)	A(0,1)	A(0,9)	A(0,1)	A(0,1)
2	A(0,2)	A(0,2)	A(0,2)	A(0,2)	A(0,2)
3	A(0,3)	A(0,3)	A(0,3)	A(0,3)	A(0,3)
4	A(0,4)	A(0,4)	A(0,4)	A(0,4)	A(0,4)
5	A(0,5)	A(0,5)	A(0,5)	A(0,5)	A(0,5)
6	A(0,6)	A(0,6)	A(0,6)	A(0,6)	A(0,6)
7	A(0,7)	A(0,7)	A(0,7)	A(0,7)	A(0,7)

Figure 8.22 Contents of an associative-mapped data cache.

Set Associative Mapped Cache



	Contents of data cache after pass:					
	$j = 3$	$j = 7$	$j = 9$	$i = 4$	$i = 2$	$i = 0$
Set 0	A(0,0)	A(0,4)	A(0,8)	A(0,4)	A(0,4)	A(0,0)
	A(0,1)	A(0,5)	A(0,9)	A(0,5)	A(0,5)	A(0,1)
	A(0,2)	A(0,6)	A(0,6)	A(0,6)	A(0,2)	A(0,2)
	A(0,3)	A(0,7)	A(0,7)	A(0,7)	A(0,3)	A(0,3)
Set 1						

Figure 8.23 Contents of a set-associative-mapped data cache.

HomeWork

A byte-addressable computer has a small data cache capable of holding eight 32-bit words. Each cache block consists of one 32-bit word. When a given program is executed, the processor reads data sequentially from the following hex addresses:

200, 204, 208, 20C, 2F4, 2F0, 200, 204, 218, 21C, 24C, 2F4

This pattern is repeated four times.

(*a*) Assume that the cache is initially empty. Show the contents of the cache at the end of each pass through the loop if a direct-mapped cache is used, and compute the hit rate

(*b*) Repeat part (*a*) for an associative-mapped cache that uses the LRU replacement algorithm.

(*c*) Repeat part (*a*) for a four-way set-associative cache..

Replacement Algorithms

- Difficult to determine which blocks to be removed
- Least Recently Used (LRU) block
- The cache controller tracks references to all blocks as computation proceeds.
- Increase / clear track counters when a hit/miss occurs

Replacement Algorithms

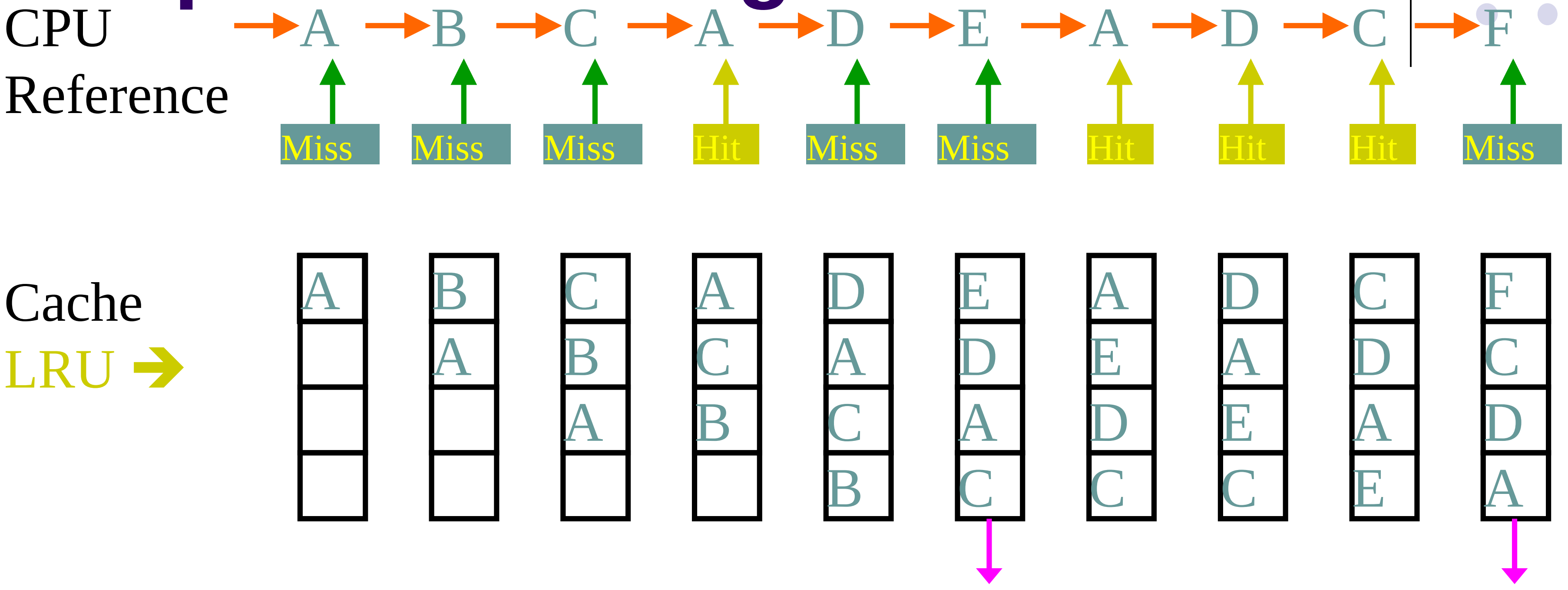
- For Associative & Set-Associative Cache

  Which location should be emptied when the cache is full and a miss occurs?

  - First In First Out (FIFO)
  - Least Recently Used (LRU)

- Distinguish an *Empty* location from a *Full* one
  - Valid Bit

Replacement Algorithms



CPU Reference	A	B	C	A	D	E	A	D	C	F
	Miss	Miss	Miss	Hit	Miss	Miss	Hit	Hit	Hit	Miss
Cache	A	B	C	A	D	E	A	D	C	F
LRU →		A	B	C	A	D	E	A	D	C
			A	B	C	A	D	E	A	D
					B	C	C	C	E	A

Hit Ratio = 4 / 10 = 0.4

- The cache controller must track references to all blocks as computation proceeds.
- Suppose it is required to track the LRU block of a four-block set in a set-associative cache.
- A 2-bit counter can be used for each block.
- When a hit occurs,the counter of the block that is referenced is set to 0. Counters with values originally lower than the referenced one are incremented by one, and all others remain unchanged.
- When a *miss* occurs and the set is not full, the counter associated with the new block loaded from the main memory is set to 0, and the values of all other counters are increased by one.

- When a miss occurs and the set is full, the block with the counter value 3 is removed, the new block is put in its place, and its counter is set to 0.
- The other three block counters are incremented by one. It can be easily verified that the counter values of occupied blocks are always distinct

Virtual Memories

Overview

- Physical main memory is not as large as the address space spanned by an address issued by the processor.

  2^{32} = 4 GB, 2^{64} = …
- When a program does not completely fit into the main memory, the parts of it not currently being executed are stored on secondary storage devices.
- Techniques that automatically move program and data blocks into the physical main memory when they are required for execution are called virtual-memory techniques.
- Virtual addresses will be translated into physical addresses.

Overview

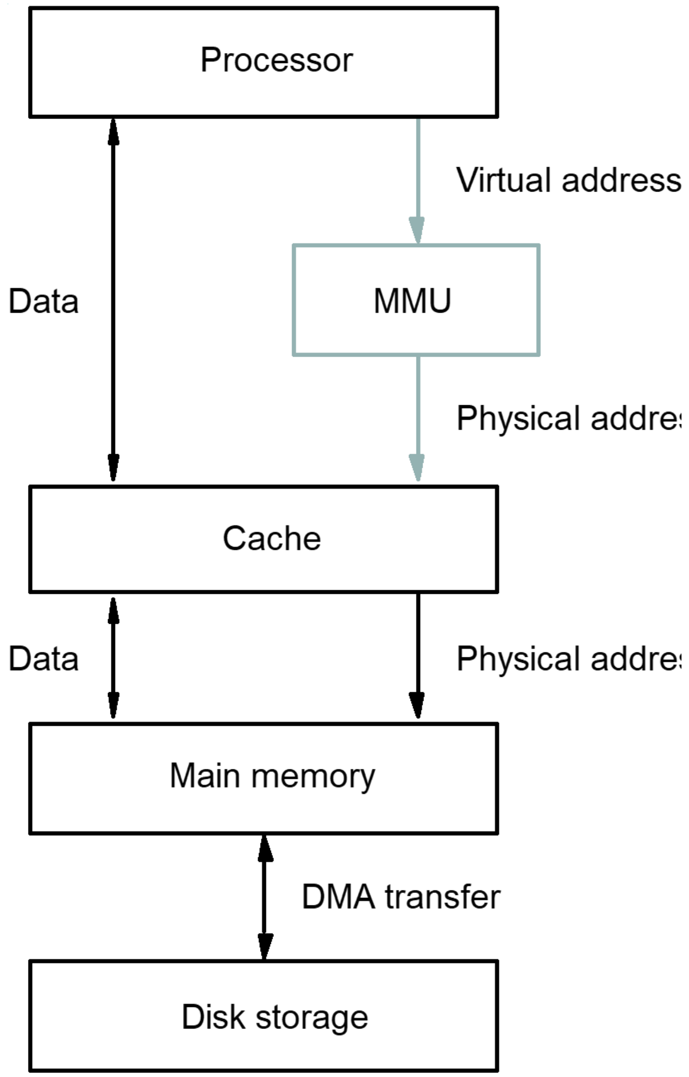


Figure 5.26. Virtual memory organizatio

Memory Management Unit

Address Translation

- All programs and data are composed of fixed-length units called pages, each of which consists of a block of words that occupy contiguous locations in the main memory.
- Page cannot be too small or too large.
- The virtual memory mechanism bridges the size and speed gaps between the main memory and secondary storage – similar to cache.

- Information about the main memory location of each page is kept in a ***page table***.
    - includes the main memory address where the page is stored and the current status of the page
        - Validity, modified
- An area in the main memory that can hold one page is called a ***page frame***.
- The starting address of the page table is kept in a ***page table base register***.
- By adding the virtual page number to the contents of this register, the address of the corresponding entry in the page table is obtained.
- The contents of this location give the starting address of the page if that page currently resides in the main memory.

Address Translation

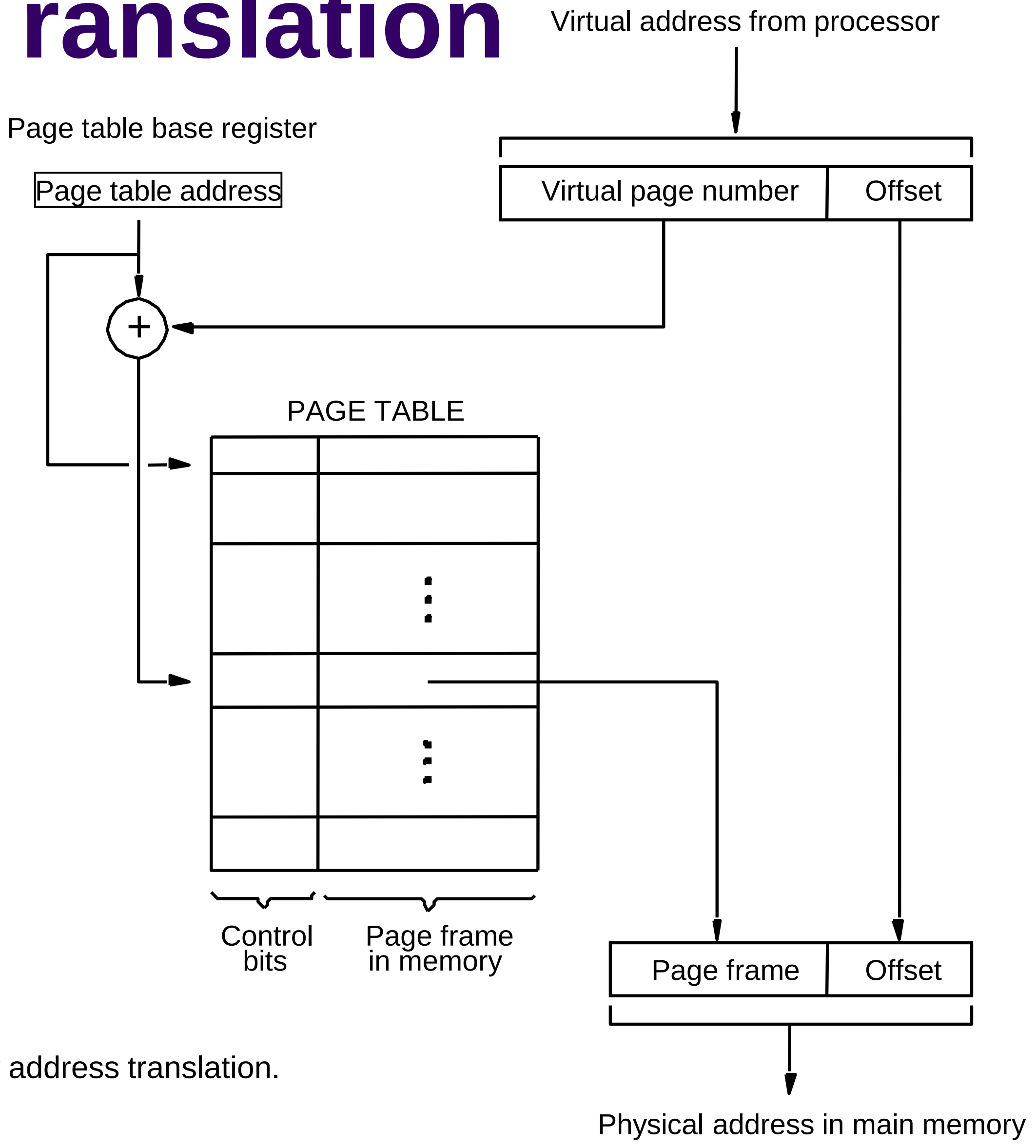


Figure 5.27. Virtual-memory address translation.

Address Translation

- The page table information is used by the MMU for every access, so it is supposed to be with the MMU.
- Since MMU is on the processor chip and the page table is rather large, only small portion of it, which consists of the page table entries that correspond to the most recently accessed pages, can be accommodated within the MMU.
- Translation Lookaside Buffer (TLB)

TLB

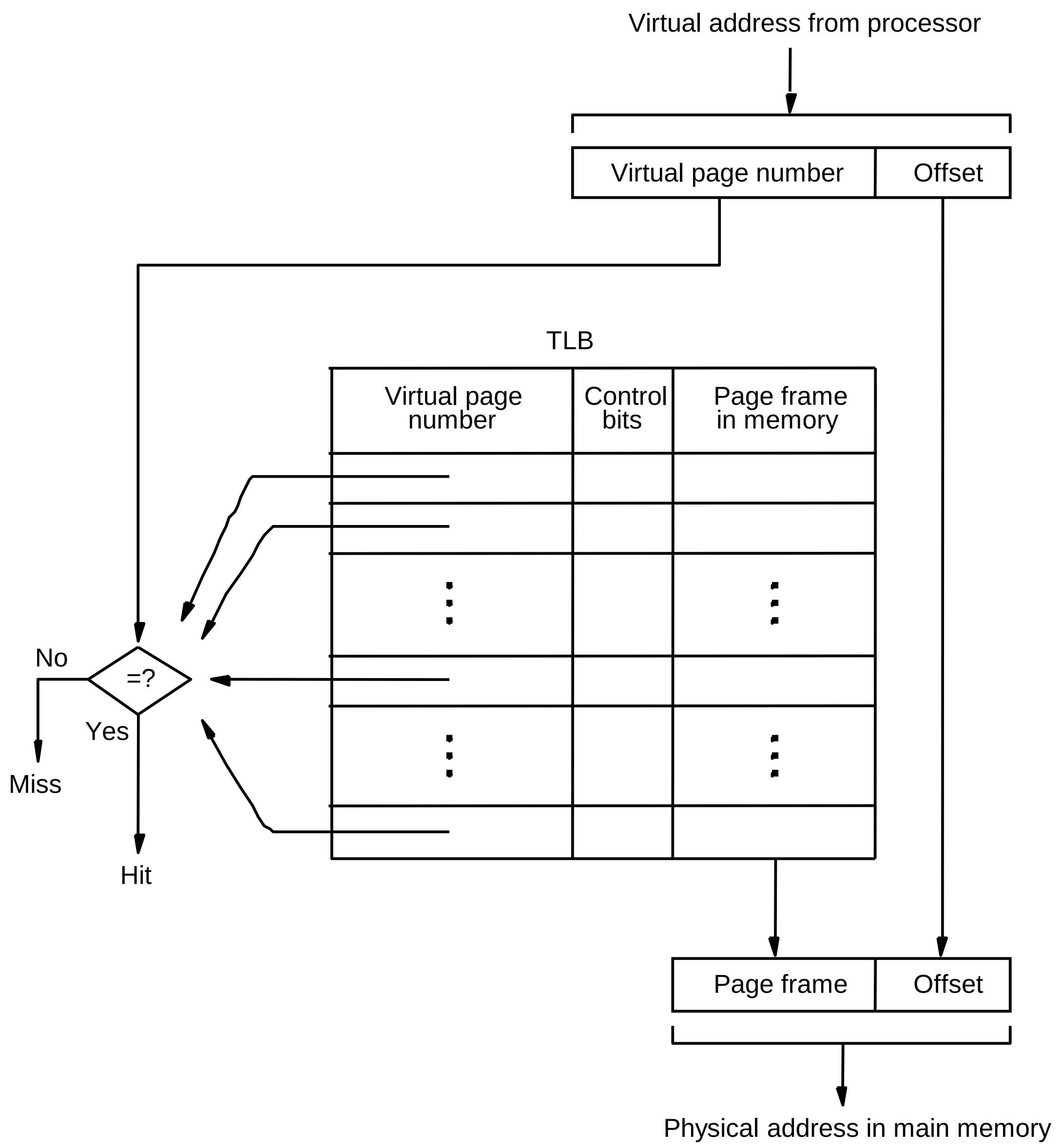


Figure 5.28. Use of an associative-mapped TLB.

TLB

- The contents of TLB must be coherent with the contents of page tables in the memory
  - When the operating system changes the contents of a page table, it must simultaneously invalidate the corresponding entries in the TLB. One of the control bits in the TLB is provided for this purpose. When an entry is invalidated, the TLB acquires the new information from the page table in the memory as part of the MMU’s normal response to access misses.
- Write-through is not suitable for virtual memory.
- Locality of reference in virtual memory

Translation procedure

- Given a virtual address, the MMU looks in the TLB for the referenced page. If the page table entry for this page is found in the TLB, the physical address is obtained immediately. If there is a miss in the TLB, then the required entry is obtained from the page table in the main memory and the TLB is updated.

Page fault

- When a program generates an access request to a page that is not in the main memory, a *page fault* is said to have occurred.
- The entire page must be brought from the disk into the memory before access can proceed. When it detects a page fault, the MMU asks the operating system to intervene by raising an exception (interrupt).

Page replacement

- If a new page is brought from the disk when the main memory is full, it must replace one of the resident pages.
- The problem of choosing which page to remove is just as critical here as it is in a cache, and the observation that programs spend most of their time in a few localized areas also applies.
- Concepts similar to the LRU replacement algorithm can be applied to page replacement, and the control bits in the page table entries can be used to record usage history

Page replacement – Contd..

- A modified page has to be written back to the disk before it is removed from the main memory. It is important to note that the write-through protocol, which is useful in the framework of cache memories, is not suitable for virtual memory. The access time of the disk is so long that it does not make sense to access it frequently to write small amounts of data

Performance Considerations

- Hit Rate and Miss Penalty
- Caches on the Processor Chip
- Other Enhancements

Hit Rate and Miss Penalty

- **hit rate -**number of hits stated as a fraction of all attempted accesses
- **miss rate-** number of misses stated as a fraction of attempted accesses
- the entire memory hierarchy would appear to the processor as a single memory unit that has **the access time of the cache on the processor chip** and the **size of the magnetic disk**
- the total access time seen by the processor when a miss occurs -***miss penalty***

Miss Penalty

- system with only one level of cache

  - the time to access a block of data in the main memory.

- Let h be the hit rate, M the miss penalty, and C the time to access information in the cache
- The average access time experienced by the processor is

$$tavg = hC + (1 - h)M$$

Example

- **C**onsider a computer that has the following parameters. Access times to the cache and the main memory are τ and 10τ , respectively. When a cache miss occurs, a block of 8 words is transferred from the main memory to the cache. It takes 10τ to transfer the first word of the block, and the remaining 7 words are transferred at the rate of one word every τ seconds. Assume that 30 percent of the instructions in a typical program perform a Read or a Write operation, which means that there are 130 memory accesses for every 100 instructions executed. Assume that the hit rates in the cache are 0.95 for instructions and 0.9 for data**.** Assume further that the miss penalty is the same for both read and write accesses
- miss penalty in this computer is given by:

$$M = \tau + 10\tau + 7\tau + \tau = 19\tau$$

$$\frac{\text{Time without cache}}{\text{Time with cache}} = \frac{130 \times 10\tau}{100(0.95\tau + 0.05 \times 19\tau) + 30(0.9\tau + 0.1 \times 19\tau)} = 4.7$$

- cache makes the memory appear almost five times faster than it really is
- Let us consider how effective the cache of this example is compared to the ideal case in which the hit rate is 100 percent

$$\frac{\text{Time for real cache}}{\text{Time for ideal cache}} = \frac{100(0.95\tau + 0.05 \times 19\tau) + 30(0.9\tau + 0.1 \times 19\tau)}{130\tau} = 2.1$$

- 100% hit rate in the cache would make the memory appear twice as fast as when realistic hit rates are used.

How can the hit rate be improved?

- make the cache larger
  - increased cost
- increase the cache block size while keeping the total cache size constant, to take advantage of spatial locality
  - the performance of a computer is affected positively by increased hit rate and negatively by increased miss penalty
  - block size should be neither too small nor too large.
  - In practice, block sizes in the range of 16 to 128 bytes are the most popular choices
- miss penalty can be reduced if the load-through approach is used when loading new blocks into the cache.

Caches on the Processor Chip

- Most processor chips include at least one L1 cache. Often there are two separate L1 caches, one for instructions and another for data.
- In high-performance processors, two levels of caches are normally used, often implemented on the processor chip.
  - separate L1 caches for instructions and data –fast- 10’s of KB
  - a larger L2 cache-slower but larger than L1- only affects the miss penalty of the L1 caches-100s of KB or MB

The average access time experienced by the processor in such a system is:

$$t_{avg} = h_1C_1 + (1 - h_1)(h_2C_2 + (1 - h_2)M)$$

where

h_1 is the hit rate in the L1 caches.

h_2 is the hit rate in the L2 cache.

C_1 is the time to access information in the L1 caches.

C_2 is the miss penalty to transfer information from the L2 cache to an L1 cache.

M is the miss penalty to transfer information from the main memory to the L2 cache.

Other Enhancements

- Each Write operation results in writing a new value into the main memory.
- If the processor must wait for the memory function to be completed, processor is slowed down by all Write requests.
- Processor does not need immediate access to the result of a Write operation; not necessary for it to wait for the Write request to be completed.
- *Write buffer* included for temporary storage of Write requests.

Write Buffer- write-through protocol (Contd..)

- Processor places each Write request into this buffer and continues execution of the next instruction.
- Sent to the main memory whenever the memory is not responding to Read requests.
- Read requests be serviced quickly, because the processor usually cannot proceed before receiving the data being read from the memory.
- These requests are given priority over Write requests

Write Buffer- write-through protocol (Contd..)

- The Write buffer may hold a number of Write requests.
- subsequent Read request may refer to data that are still in the Write buffer.
- To ensure correct operation, the addresses of data to be read from the memory are always compared with the addresses of the data in the Write buffer.
- In the case of a match, the data in the Write buffer are used.

Write Buffer- write-back protocol

- Write commands issued by the processor are performed on the word in the cache.
- When a new block of data is to be brought into the cache as a result of a Read miss, it may replace an existing block that has some dirty data.
- The dirty block has to be written into the main memory.
- If the required write-back is performed first, then the processor has to wait for this operation to be completed before the new block is read into the cache.

Write Buffer- write-back protocol(Contd..)

- It is more prudent to read the new block first.
- The dirty block being ejected from the cache is temporarily stored in the Write buffer and held there while the new block is being read.
- Afterwards, the contents of the buffer are written into the main memory.
- Write buffer also works well for the write-back protocol.

Prefetching

- new data are brought into the cache when they are first needed.
- Following a Read miss, the processor has to pause until the new data arrive, thus incurring a miss penalty.
- To avoid stalling the processor, it is possible to prefetch the data into the cache before they are needed
- Through
  - software -prefetch instruction
  - hardware- using circuitry that attempts to discover a pattern in memory references and prefetches data according to this pattern
- Prefetch-Executing this instruction causes the addressed data to be loaded into the cache, as in the case of a Read miss.

Prefetching-Contd..

- The hope is that prefetching will take place while the processor is busy executing instructions that do not result in a Read miss, thus allowing accesses to the main memory to be overlapped with computation in the processor
- Inserted either by programmer or compiler
- Overhead
  - Increases length of program
  - Data may not be used by instructions that follow-if the prefetched data are ejected from the cache by a Read miss involving other data

Lockup-Free Cache

- Software prefetching does not work well if the action of prefetching stops other accesses to the cache until the prefetch is completed
- While servicing a miss, the cache is said to be locked.
- modify the basic cache structure to allow the processor to access the cache while a miss is being serviced
- A cache that can support multiple outstanding misses is called ***lockup-free***.
- cache must include circuitry that keeps track of all outstanding misses

Problem

- Suppose that a computer has a processor with two L1 caches, one for instructions and one for data, and an L2 cache. Let τ be the access time for the two L1 caches. The miss penalties are approximately 15τ for transferring a block from L2 to L1, and 100τ for transferring a block from the main memory to L2. Assume that the hit rates are the same for instructions and data and that the hit rates in the L1 and L2 caches are 0.96 and 0.80, respectively.

(a)What fraction of accesses miss in both the L1 and L2 caches, thus requiring access to the main memory?

(*b*) What is the average access time as seen by the processor?

(*c*) Suppose that the L2 cache has an ideal hit rate of 1. By what factor would this reduce

the average memory access time as seen by the processor?

(*d*) Consider the following change to the memory hierarchy. The L2 cache is removed and the size of the L1 caches is increased so that their miss rate is cut in half. What is the average memory access time as seen by the processor in this case?

(*a*) The fraction of memory accesses that miss in both the L1 and L2 caches is

$$(1 - h_1)(1 - h_2) = (1 - 0.96)(1 - 0.80) = 0.008$$

(*b*) The average memory access time using two cache levels is

$$\begin{aligned} t_{avg} &= 0.96\tau + 0.04(0.80 \times 15\tau + 0.20 \times 100\tau) \\ &= 2.24\tau \end{aligned}$$

(*c*) With no misses in the L2 cache, we get:

$$t_{avg}(\text{ideal}) = 0.96\tau + 0.04 \times 15\tau = 1.56\tau$$

Therefore,

$$\frac{t_{avg}(\text{actual})}{t_{avg}(\text{ideal})} = \frac{2.24\tau}{1.56\tau} = 1.44$$

(*d*) With larger L1 caches and the L2 cache removed, the access time is

$$t_{avg} = 0.98\tau + 0.02 \times 100\tau = 2.98\tau$$

Reference

Carl Hamacher,Zvonko Vranesic, Safwat Zaky and Naraig Manjikian, “Computer Organization and Embedded Systems”, Sixth Edition, McGraw Hill