COMPUTER ORGANIZATION AND ARCHITECTURE

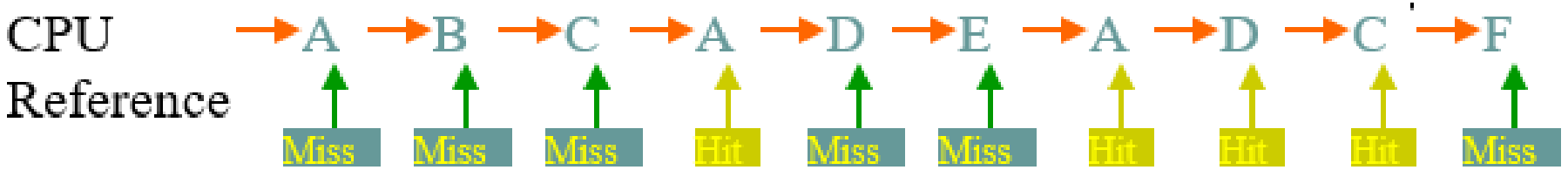
Course Code : CSE 2151

Credits : 04

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



A	A	A	**A**	A	A	**A**	A	A	A
	B	B	B	B	E	E	E	E	F
		C	C	C	C	C	C	**C**	C
				D	D	D	**D**	D	D

Hit Ratio = 4 / 10 = 0.4

LRU ALGORITHM

- 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

LRU ALGORITHM

- Assume counter values as shown

00	Block0
01	Block1
10	Block2
11	Block3

- Suppose hit occurs for block 2

01	Block0
10	Block1
00	Block2
11	Block3

LRU ALGORITHM

- 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.

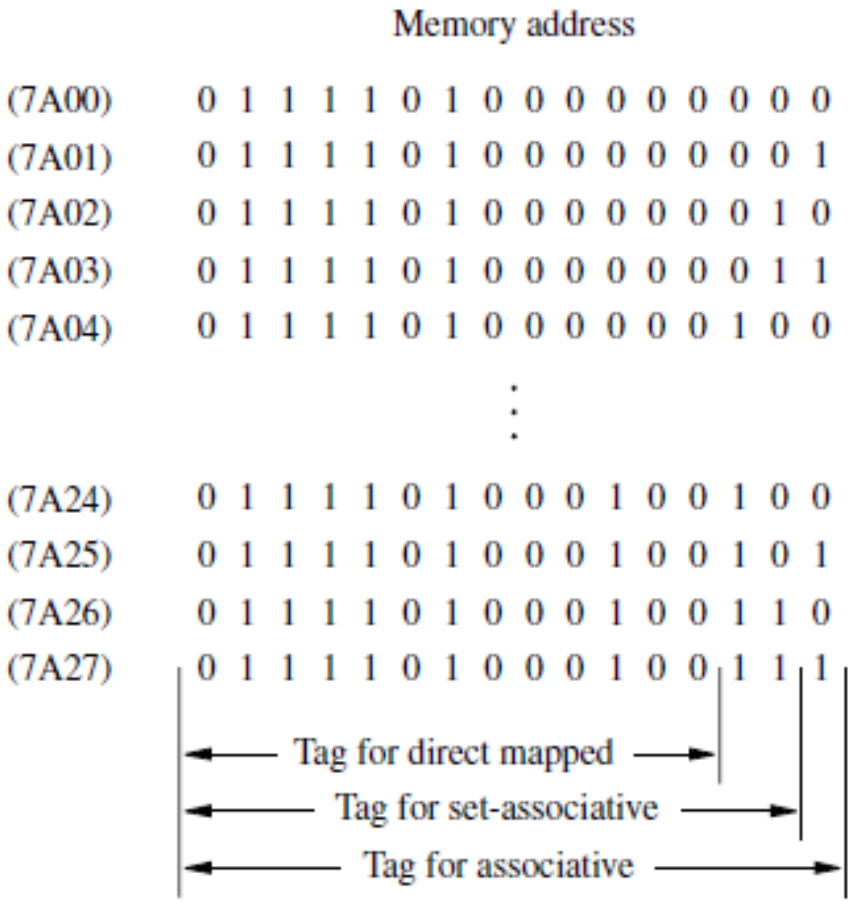
$$A(0,i) \leftarrow \frac{A(0,i)}{\left(\sum_{j=0}^{9} A(0,j)\right)/10} \qquad \text{for } i = 0, 1, \ldots, 9$$

```
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.

LRU ALGORITHM

- No.of words in each block =1 (2^0=1)
  - Hence 0 bits to identify words within the block
- Direct mapped:
  - 8 blocks in cache: 3 bits to represent
- Set-associative:
  - 1 bit to identify set 0 or 1
- Associative:
  - All bits are considered tag



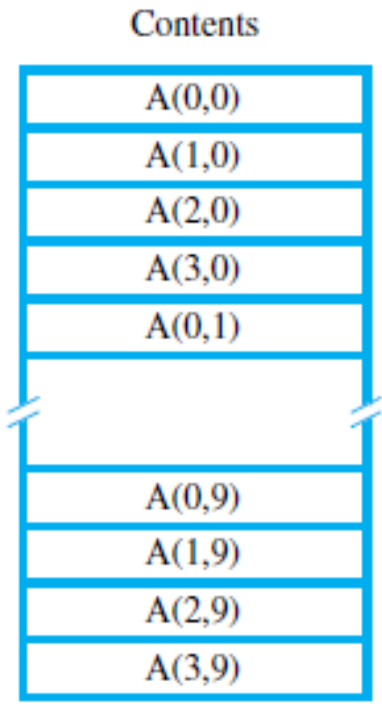


Figure 8.19 An array stored in the main memory.

DIRECT MAPPED CACHE

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

Block position	Contents of data cache after pass:				
	$j = 7$	$j = 8$	$j = 9$	$i = 1$	$i = 0$
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

VIRTUAL MEMORIES

- 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.

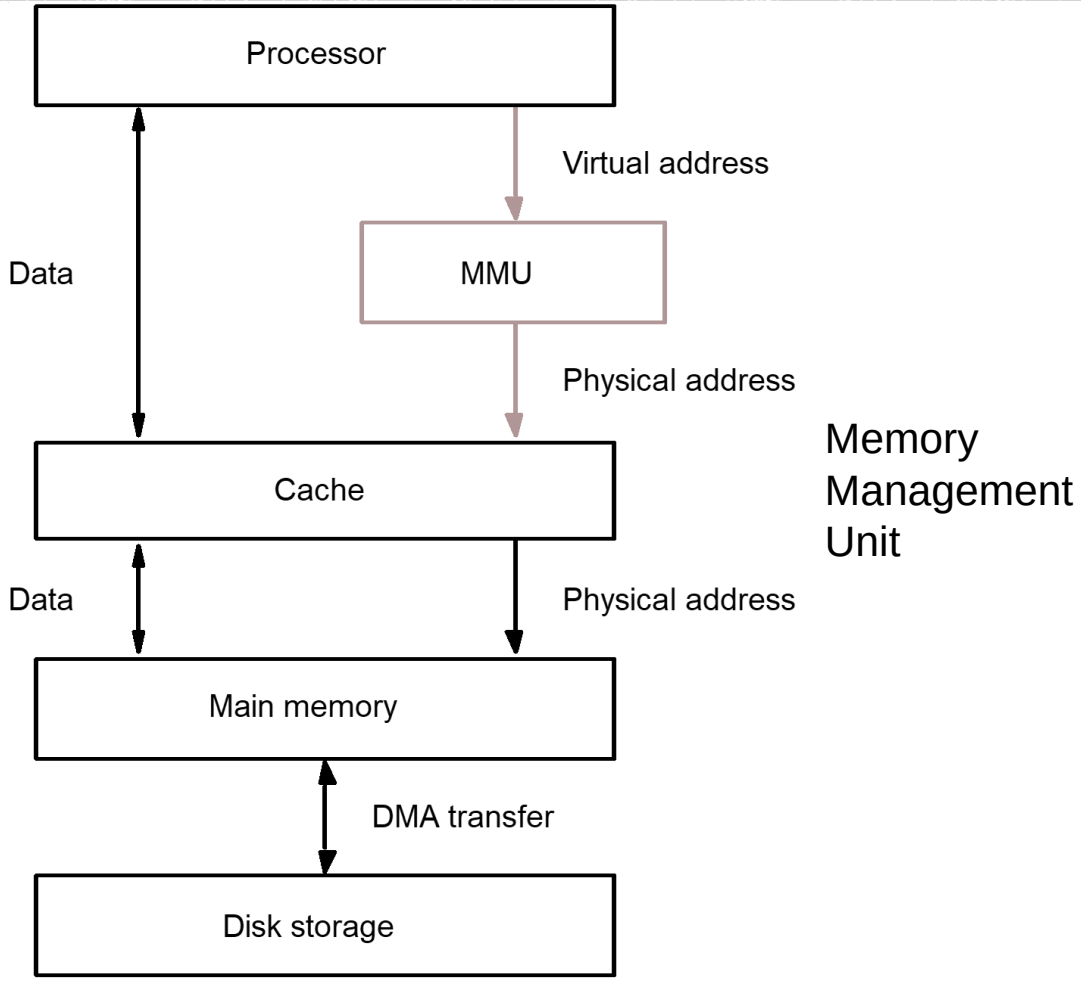


Figure 5.26. Virtual memory organization.

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 – like 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

- 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)

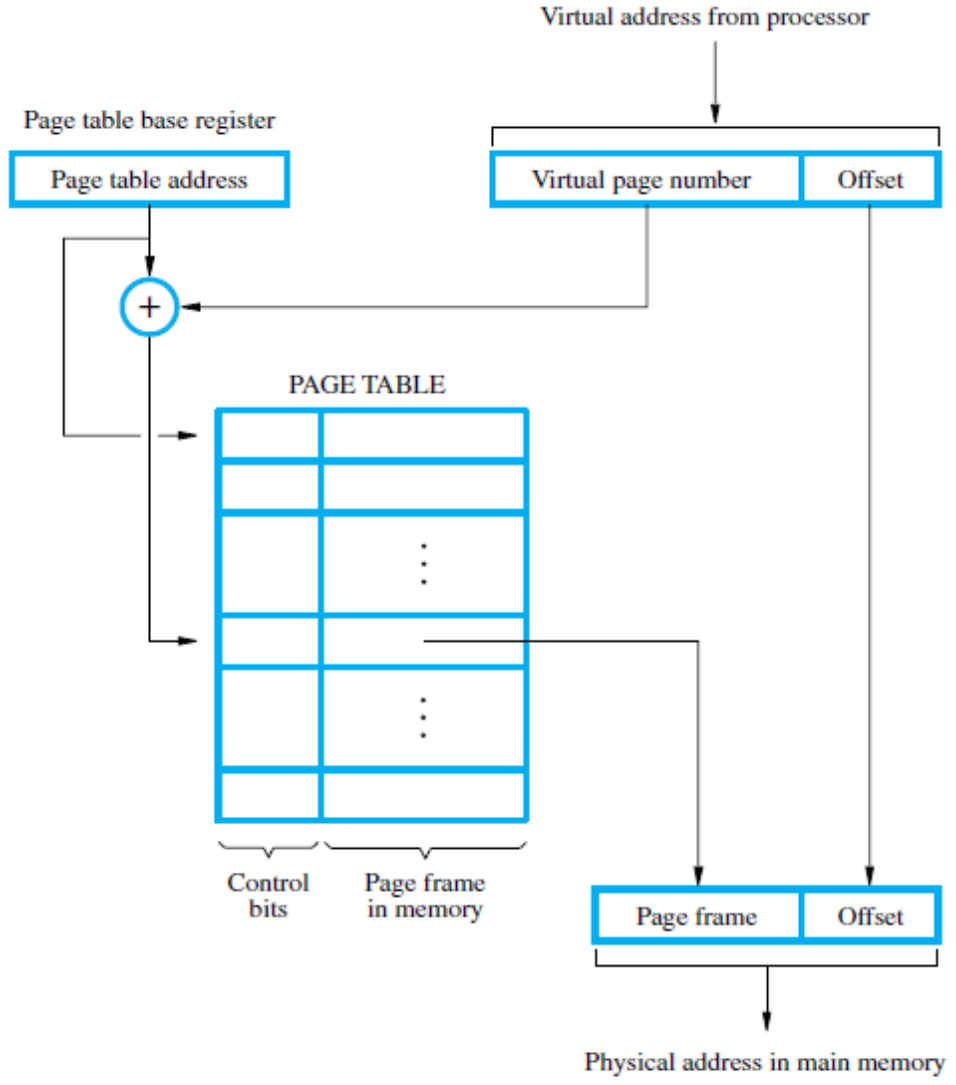


Figure 8.25 Virtual-memory address translation.

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.
- 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

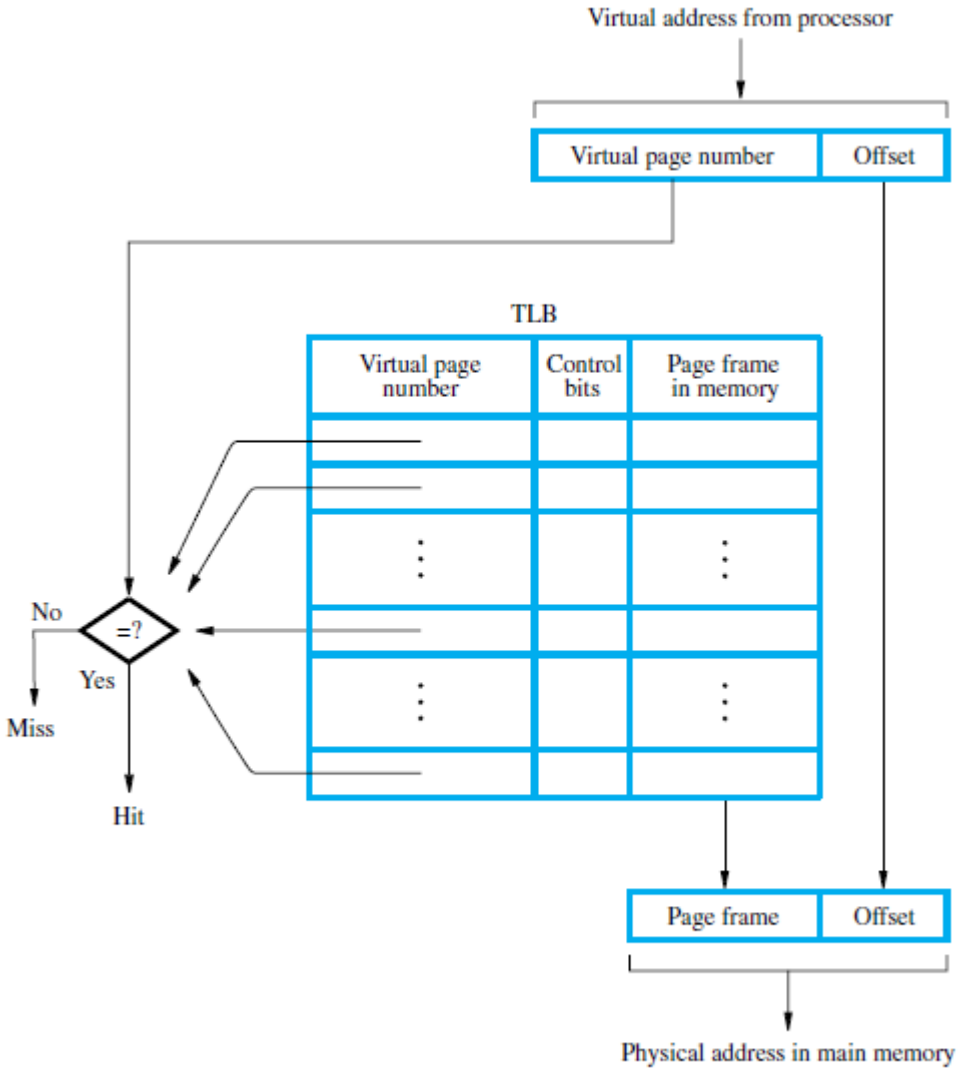


Figure 8.26 Use of an associative-mapped TLB.

PAGE FAULT AND PAGE REPLACEMENT

- 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
  - 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

EXERCISE PROBLEM

- 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.

TOPICS COVERED FROM

- Textbook 1:
  - Chapter 8: 8.6.2, 8.6.3, 8.8