SE assignment **Univ. Rennes 1**

Virtual memory and address translation

Virtual memory allows our computer machines to benefit of a very large memory space to be exploited by our running programs. The RAM memory, which is actually much smaller than virtual memory, is supposed to contain only the data (organized in *pages*) that the computer machine needs to access at a given time. All the other data are temporarily stored in the hard drive, and they are retrieved only when a program asks for having access to them.

In this assignment, we will study in details some of the procedures that need to be executed in the computer machine for an efficient management of the virtual memory. We will mainly focus on the so-called *mapping*, which allows us to translate every virtual address of the virtual memory, in a physical address of the RAM.

It is important to point out that, in modern computer machines, the mapping is managed at hardware level by a dedicated circuit named "Translation Lookaside Buffer" (TLB), which is (physically) placed next to the CPU. In this assignment, however, we will pretend that it is rather our task to take care of the different necessary actions, in order to better understand them.

We will study a virtual memory with the following properties:

- 14 bits are used for the virtual addresses;
- 12 bits are used for the physical addresses;
- 64 bytes are used for the page.

It is supposed that a 4-way associative mapping is implemented for the address translations.

Please make reference to the page table available in the last page of this document (only the information necessary to perform our assignment is provided).

Question 1

Since the size of a page is 64 bytes, how many bits are necessary to guarantee the existence of a unique address for every byte in the page? As a consequence, how many bits are left in every (virtual and physical) address that actually represent the address of the page? It is in fact important to point out that the virtual and physical addresses that are contained in these tables concern *pages*, and not single memory locations (such as the address to an integer or a character variable).

Question 2

How many possible virtual page addresses we have in our page table? Same question for the physical page addresses.

Question 3

Since we have opted for a 4-way associative table, how many bits do we need to represent the index of the virtual page address, and how many bits are necessary to represent the tag? The index and the tag are related to two different methods the perform the address translations. Do you remember which ones?

Question 4

Consider the virtual address 10111010001100 for a memory location. By considering the virtual address of the page, and the fact that this address is valid in the table, construct the physical address for the memory location.

Question 5

Consider the virtual page address 01110110. This address is not valid in the page table: when the processor will try to access to its content, there will be a "page fault". Is it necessary to free some space in the RAM to load these data? Once the data are loaded, how to update the page table?

Question 6

Now consider the virtual page address 11010010. It's again a page fault. What is the difference with the situation in the previous question? Please update the page table.

Question 7

Now consider the virtual page address 10111100. Page fault: is there enough space in the RAM to load this page? Should we select a victim page? If yes, use the FIFO method (FIFO = *First-In First-Out*). Suppose, for simplicity, that the time values in the original page table have not changed since the beginning of our exercise: you can suppose that things are happening so fast in the computer machine that these values, let's say represented in milliseconds, have not had *the time* to change yet...

Question 8

Now consider the virtual page address 10111001. In case a victim page needs to be selected, use now the LRU method (LRU = *Least Recently Used*).

Corrections

Question 1

Since the size of a page is 64 bytes, how many bits are necessary to guarantee the existence of a unique address for every byte in the page? As a consequence, how many bits are left in every (virtual and physical) address that actually represent the address of the page? It is in fact important to point out that the virtual and physical addresses that are contained in these tables concern *pages*, and not single memory locations (such as the address to an integer or a character variable).

> 6 bits are necessary to give a unique access to the 64 bytes forming the pages. Therefore, 8 out of the 14 bits for the virtual addresses can be employed for the page address, while the remaining 6 bits can be used as for an offset on the 64 bytes. In the RAM, 6 bits are used for the page address, and 6 bits are used for the offset.

Question 2

How many possible virtual page addresses we have in our page table? Same question for the physical page addresses.

> We have 2^8 potential virtual page addresses, and 2^6 physical page addresses.

Question 3

Since we have opted for a 4-way associative table, how many bits do we need to represent the index of the virtual page address, and how many bits are necessary to represent the tag? The index and the tag are related to two different methods the perform the address translations. Do you remember which ones?

> In a 4-way associative table, we can associate 4 different physical addresses to virtual addresses sharing the same *index*. In order to implement this flexibility, we need to use 2 bits of the physical address for the address *tag*, which represents the 4 possible memory locations in the RAM. Therefore, only 4 out of the 8 initial bits in the virtual address are left for the address index. This index is computed by simply cutting off the extra bits of the virtual page address (in our example, only the first 4 bits of the virtual page address are considered). The tag can take any of the 4 possible, and not used yet, values (we can suppose that the arithmetic ordering is used for the assignment of the tag).

Question 4

Consider the virtual address 10111010001100 for a memory location. By considering the virtual address of the page, and the fact that this address is valid in the table, construct the physical address for the memory location.

> The address is for a memory location, and it is composed by 14 bits: the first 8 bits represent the virtual address of the page, while the remaining 6 bits represent the offset to identify the memory location inside the page. Since the size of the page is the same in virtual and physical memories, the offset can be reused without any modification in the physical address. The main question is therefore how to translate the virtual page address in the physical page address. When looking at current page table (see below), we can see that the virtual address 10111010 is valid, and that the corresponding physical address is 101101 (1011 is the index, 01 is the tag). Therefore the physical address of the memory location is 101101001100.

Question 5

Consider the virtual page address 01110110. This address is not valid in the page table: when the processor will try to access to its content, there will be a "page fault". Is it necessary to free some space in the RAM to load these data? Once the data are loaded, how to update the page table?

> No, it is not necessary to free space in the RAM to load the requested page (because all tags of corresponding index in physical address are currently available). See the updated page table below (in blue the added address).

virtual page address	valid	4-way association	physical page address	time since in RAM	time since accessed
00111010	1	00	001100	6	5
01110110	**1**	**00**	**011100**	**0**	**0**
10111000	1	00	101100	7	1
10111001	0				
10111010	1	01	101101	1	3
10111011	1	10	101110	3	4
10111100	0				
10111101	1	11	101111	2	8
10111110	0				
11010001	1	00	110100	5	2
11010010	0				
11011001	1	01	110101	9	1
11110000	0				

Question 6

Now consider the virtual page address 11010010. It's again a page fault. What is the difference with the situation in the previous question? Please update the page table.

> The difference with the previous question is that now the index related to this virtual address is already present in the page table. When constructing the physical page address, therefore, we need to consider the fact that we cannot use tags that are already assigned to other valid pages. Since the physical address 110100 is already in use (index is 1101, tags are 00 and 01), we can now associate the address 110110 to this new page (the index is the same, but the tag is 10). See the updated page tables below.

virtual page address	valid	4-way association	physical page address	time since in RAM	time since accessed
00111010	1	00	001100	6	5
01110110	0	00	011100	0	0
10111000	1	00	101100	7	1
10111001	0				
10111010	1	01	101101	1	3
10111011	1	10	101110	3	4
10111100	0				
10111101	1	11	101111	2	8
10111110	0				
11010001	1	00	110100	5	2
11010010	**1**	**10**	**110110**	**0**	**0**
11011001	1	01	110101	9	1
11110000	0				

Question 7

Now consider the virtual page address 10111100. Page fault: is there enough space in the RAM to load this page? Should we select a victim page? If yes, use the FIFO method (FIFO = *First-In First-Out*). Suppose, for simplicity, that the time values in the original page table have not changed since the beginning of our exercise: you can suppose that things are happening so fast in the computer machine that these values, let's say represented in milliseconds, have not had *the time* to change yet...

> The index is 1011. It already appears several times in this page table, and the 4 tags are already in use. For this reason, it is in fact necessary to select a victim page: we select a page to be removed from the RAM (and as a consequence its physical address is removed from the page table); then, the requested page will take the place, in the physical memory, of the victim page (the new page will have the physical address of the victim page). See the updated page table for the details: the victim page, in red, selected by the FIFO method is the one that has spent longer time in the RAM.

virtual page address	valid	4-way association	physical page address	time since in RAM	time since accessed
00111010	1	00	001100	6	5
01110110	0	00	011100	0	0
10111000	**0**	**00**	**101100**	**7**	**1**
10111001	0				
10111010	1	01	101101	1	3
10111011	1	10	101110	3	4
10111100	**1**	**00**	**101100**	**0**	**0**
10111101	1	11	101111	2	8
10111110	0				
11010001	1	00	110100	5	2
11010010	1	01	110101	0	0
11011001	1	01	110101	9	1
11110000	0				

Question 8

Now consider the virtual page address 10111001. In case a victim page needs to be selected, use now the LRU method (LRU = *Least Recently Used*).

> The index is again 1011. As we could verify in the previous question, all tags for this index are already in use for 4 different valid addresses. It is necessary therefore to select another victim page before loading the requested page. This time, we will use the LRU method to select the victim page: it's the page that is not used since longer time. See the updated page table for the details.

virtual page address	valid	4-way association	physical page address	time since in RAM	time since accessed
00111010	1	00	001100	6	5
01110110	0	00	011100	0	0
10111000	0				
10111001	**1**	**11**	**101111**	**0**	**0**
10111010	1	01	101101	1	3
10111011	1	10	101110	3	4
10111100	1	00	101100	0	0
10111101	**0**	**11**	**101111**	**2**	**8**
10111110	0				
11010001	1	00	110100	5	2
11010010	1	01	110101	0	0
11011001	1	01	110101	9	1
11110000	0				