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*).

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				
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				