Lab 4: Multi-Level Page Table

The goal of this lab assignment is to design and implement a 2-level page table, which is an essential component of virtual memory.

In this assignment, you will work on implementing a 2-level page table. The assignment assumes a physical address space of **12-bits** and a **14-bit** virtual address space. Each page table fits into a single page, and each page table entry is **4 bytes**. The page size is **64B** and the memory is byte addressable.

Virtual address:

The virtual address size of this design will be **14** bits. The following depicts the virtual address layout:

13		0
4-bit outer page table	4-bit inner page table	6 bit offset

Page table structure:

The page/frame size is **64B**. The size of a page table entry is **4B**. For both levels, bit 0 is used as the valid bit. In the outer page table, the top **12** bits contain the physical address of the beginning of the next level page table. Bits **1-19** are unused and must be set to zero. The following depicts the PTE format for the outer page table.
To access the outer page table, the **4-bit** outer page table value in the virtual address should be added to the value stored in the page table register. You should shift this value by 2 to point to the beginning of an inner page table (Note: The memory is byte addressable). We don't have any negative numbers here, so you just need to pad with zero.

31 20	19 1	0
PA#	Not Used	Valid

In the inner page table, the top **6** bits represent the frame number, and bits **1-25** are unused. The following table depicts the PTE format for the inner page table. To access this level of the page table, the **4-bit** inner page table value in virtual address should be added to the frame number coming from the outer page table. Same as the outer level page table, the 4-bit should be shifted by 2.

31 26	25 1	0
frame#	Not Used	Valid

Physical addresses:

The physical address size is **12** bits.

Page table register file:

This special register points to the start of the outer page table. It is a **12-bit** register.

Number of page table levels:

To successfully complete the assignment, you must calculate the size of the inner page table yourself.

Implementation requirements:

We have provided the skeleton code that initializes the 2-level page table and the page table register file. Your task is to implement a function that can index the page table and print out the corresponding physical addresses (12-bit) and the value stored in the physical memory (32-bit) as a hex representation.

Your function should correctly translate virtual addresses to physical addresses using the 2-level page table structure. If the page does not contain a valid frame number (the valid bit is "0"), your function should print out "0" in the output, otherwise it should print out "1".The same applies to both outer and inner level page tables. In case the physical address cannot be retrieved, print **0x000** and **0x00000000** for the physical address and the memory value, respectively.

Your output file should have the following format for each memory request:

0/1, 0/1, 0xPhysical_address(12-bits), 0xMemory_value(32-bits)

Your code will be checked with a sequence of virtual addresses. You should save the above information for each request in a file named **"pt_results.txt."**.
We have provided an example of queries to the page table (**"pt_requests.txt"**), initialization of physical memory (**"pt_initialize.txt"**) and paget table register (**"PTBR.txt"**), and the output file (**"pt_results.txt"**) in the "expected_results" directory, which shows the exact format of the output file.

The "pt_requests.txt" and "PTBR.txt" should be given as an input. Your code should be executed as follows:

>>> ./PageTable pt_requests.txt PTBR.txt

Your Assignment:

1. We have provided the skeleton code in the file PageTable.cpp. Implement the functions where you see "TODO: implement".
    a. A Makefile has been provided for you to compile the source code (do not modify the Makefile)
    b. We will be compiling your code with: `g++ version 11.3.0`. **Please make sure that your code is compatible with g++**
    c. You can compile your design by typing "make" which will create an executable called "PageTable". Run the executable by calling: "./PageTable".
    d. Calling the binary "./PageTable" expects the "pt_initialize.txt" file in the same directory and produces one output file name "pt_results.txt".

   e. We have provided “pt_initialize.txt”, “PTBR.txt”, and “pt_requests.txt” files containing initialization of physical memory and page table register file and a sample sequence of queries to the page table. These files must be in the same directory as the source code.
   f. We have provided the correct output results for the provided test case in the “expected_results” directory. Make sure you pass this test case before submitting your code. However, we encourage you to write your own sequence of requests and check your design.
2. You will be submitting your assignments on Gradescope. **You must upload ONLY one file on Gradescope: “PageTable.cpp”. Do not zip the file or upload any other file.**

Due Date:

Friday Nov 17, 2023, 23:59 Eastern Time.