CS431 Operating Systems: Winter 2018 **Due: 3/04/2018 by 11:59PM**

Project 2
Virtual Memory Simulator
(100 points)

Introduction:
For project 2 you will implement a virtual memory simulator. This simulator consists of several important parts: the CPU which contains the memory management unit and the TLB cache, the virtual page table, physical memory, and the operating system. This system must be modular, that is, the aforementioned parts must be represented by at least one class each. **You are allowed to use Java or C++ to implement this simulator.**

Implementation:
Data Structures:
You must use fixed-size arrays to simulate the page table, the TLB, the page, and physical memory.

- Physical memory will be a two-dimensional array to simulate the page-frame # and the page of byte-addressable, byte-sized data. For instance, if physical memory can hold 16 pages of data and each page is 1kB, then you would have a 2d array like: ram[16][1024]. Read further to determine what values you must use in your simulator. In each byte of ram you will store an integer value (yes, an int is typically larger than a byte but this will make the project simpler).
- The page table will be an array of PageTableEntries (so the page table will be a one-dimensional array). You will use the virtual page number (V-Page#) as the index to the elements of this array.

  Table entry for the virtual page table:

V	R	D	PageFrame#

- The TLB will be a one-dimensional array of TlbEntries. A TlbEntry consists of a virtual page number and a frame number. The TLB is small and must be scanned on every lookup. The arrays used to implement the page table and TLB will be arrays of data structures that represent the tables' entries.

  Table entry for the TLB:

V-Page#	V	R	D	PageFrame#

Data Files:
Included with the project specification is a file titled, "CS431_project2_test_and_page_files.zip". This zip file contains the test files and page files that will simulate the various reads and writes to addresses by a running process (test files), and the process' pages of data that the OS must load from the hard drive and put into memory. If a write to memory occurs and the OS needs to evict the dirty page, then the OS must write the page back to the file on the hard disk.

The test file format is:

```
0
1C85
1
A1B5
8517174
...
```

Where the first value represents whether it is a read (0) or write (1), The second value is the address to be read from or written to. If the action is a write (1) then there will be a third value which represents the integer to be written to memory.

The page files hold the data elements from the processes virtual memory.

The Simulated Computer:

- The CPU address width is 16 bits
- Physical memory's address width is 12 bits
- The page offset is 8 bits.
- The TLB contains 8 entries.
- The OS uses the clock algorithm for page replacement
- The MMU uses FIFO for replacement algorithm
- The OS resets the r-bit every 20 instructions.

Note: you are not permitted to use any containers or data structures provided by your chosen language other than simple arrays (i.e., no arraylists). You must implement the clock replacement algorithm's data structure as a circular linked list.

Execution:

Your program will accept as a command line argument the test file path to a text file populated with virtual memory addresses (in hex and also provided with this project) that are used to simulate memory accesses called by a process. Your CPU will read them, hand them to the MMU for fetching or writing. If the address is preceded by a zero, then the MMU should only read and output the value. If the address is preceded by a one, then the address will be followed by a decimal value that needs to be written to physical memory. In the latter case, the page containing the data must have it's dirty bit set by the MMU in both the TLB and the page table. This page must be written back to disk if your page replacement algorithm decides to replace the file.

For each test file that you run, you should use an original copy of the page files that are available with the project. You do not want altered page files of a previous run to be used. Keep the originals in a safe place and overwrite the working set for each simulation. This should be done programmatically from within your simulator, not manually. You should use a file copy facility of your chosen language, not an operating system call. You cannot assume that the operating system of the grader is the same as yours.

The page files hold the simulated data of the process' virtual memory. Each filename is titled with the page number that it represents.

Output:

Your program will output the following information in a CSV file (with appropriate headers):

The address, Read or write (0 or 1), The value read or written, Soft miss (0 = false, 1 = true), Hard miss (0 = false, 1 = true), A hit (0 = false, 1 = true), page number of the evicted page, was that page's dirty bit set.

The header of the CSV should be:

Address, r/w, value, soft, hard, hit, evicted_pg#, dirty_evicted_page

There should be one csv for every test file.

More on Page Replacement:

We will keep the page replacement algorithm simple. Please implement the clock algorithm for page replacement. When a hard miss occurs, it's the job of the OS to replace the page and write the evicted page back to the drive if the dirty bit is set. The CPU should trap to the OS when hard misses occur.

The MMU is responsible to set the r-bit and the d-bit; the OS is responsible for unsetting them. The MMU should set the r-bit on a read or write of data in both the TLB and the page table. It should set the dirty bit on a write. The OS should unset the r-bits of all table entries after the CPU processes twenty instructions; it should reset the d-bit after a page has been written back to the disk.

When the MMU encounters a soft or hard miss it will need to overwrite a record in the TLB with the entry of the page being called for. You can use the FIFO replacement algorithm for this. The MMU should also set the r-bit in the page table when it needs to access it on a lookup.

Teams:
You are allowed to work in teams of up to four students. **Each student must submit a copy of the project.** The project must contain a statement listing the individuals on your team and whether or not they contributed equally to the project (a word document will be provided).

A word of advice, choose your teammates wisely. If you cannot find someone that you trust to treat this project with the same enthusiasm as you do, then work on the project by yourself. It sounds like a lot but it is not that difficult if you approach the problem in an organized fashion and start work on it immediately.

If you choose to work in a team, your first order of business should be to meet and discuss the design of your program and the responsibilities of the members. You should set milestones for your work so that you hold each other accountable and you do not fall behind. Once you form teams, or if you are working solo, one member of each team should email to me a list of your team members.

Turn in (via Blackboard):
1. Your source code; include a README file to explain how to compile and run your program. You cannot assume that the grader will use your IDE so give instructions on how to compile from the command line.
2. Your output files
3. Compress your submission into a single file.
4. If you are on team, include a completed "Team Member Scorecard" form.

Grading:
You will be graded on the modularity of your design and its correctness. The grader will test your program for correctness with a separate input file. Do not alter the provided page files or test files to accommodate your program. Your design must demonstrate the responsibilities of the various components of this system as was discussed in class.