Ian Spiegel, Chris Smith
Project 3 Report
CSC345-01

In this project, we implemented a method to correctly read in logical addresses from addresses.txt, correctly translate those input addresses into a physical address, correctly retrieve the values stored values in the physical address, implemented FIFO based TLB update, correctly counted the number of page faults in the txt file, correctly counted the number of TLB hits in the txt file, and implemented FIFO page replacement in main-pr.c.

The way we implemented correctly reading in logical addresses from addresses.txt, is by using fopen to open the file, checking if the file exists, and then running the opened file through a while loop that collects the logical address, physical address and value by looping functions pager, TLBFIFOPR, and BSreader. These functions check the data we got from the txt file and use it to calculate what the values for each logical address should be as well as their physical addresses.

The way we translated the input addresses into physical addresses is through the pager function in our code. This function finds the page number and offset from the logical address and tries to find a matching page number in the TLBpagenumber buffer, if one is found, it's frame is extracted from the buffer and the TLB hit counter is incremented. If not, we go through the function Backing storage reader (BSreader). Then the logical address will go through the TLB FIFO page updater which will figure out the value. After this, the values are printed on their respective txt files.

The Pager function will find the values stored inside the physical address by keeping a 2d array of physical addresses inside the variable PM. We then collect the offset and frame number to find the value of a certain physical address. Each physical address value can be found by using value=PM[framenumber][offsetvalue]. These values are already found from the previous code in the pager function and help from the TLB updating function.

TLBFIFOPR is the function used to update the TLB buffers for page and frame based in FIFO techniques. This function uses page number and frame number to check through the TLB and organize the frames and pages so no overflow, leaking, or overall corruption occurs. If TLB has room, we place the page and frame inside their respective TLB, if not, we must move everything over to make room for the new frame and page. After moving everything over, the new page and frame will be inserted on the end of their respective TLB.

While running through Pager, every time the frame number would equal -1, there would be a pageFault and BSreader will be called (the backing store). So, we keep an if statement to check for whenever the frame is -1 to count pageFaults while the code is running through each address.

Whenever the TLBPageNumber index equals the page number of the TLB buffer, then there is a TLBHit. Knowing this, we implemented a for loop to check the entire TLB for if there is a hit for every new address being added in. If there is a hit, the counter for TLB hits is incremented and printed at the end when everything is read in from addresses.txt.

For the final part (creating main-pr.c), we check if there are any times the frame number is not equal to the page number for all frames numbers. If all frames are full, filling up to frame 128, and the page request is not found in the frame array, we replace the oldest page in our array that stores all the frame numbers, TLBPageNumber. Since changing this array changes our results for the amount of TLBHits, there is a second array called temp[TLBS] which acts as a copy of TLBPageNumber before we commit changes to TLBPageNumber. This will ensure that changing the array TLBPageNumber will not affect our results for TLBHits.