CSC 415 - Virtual Memory Manager
Project 4

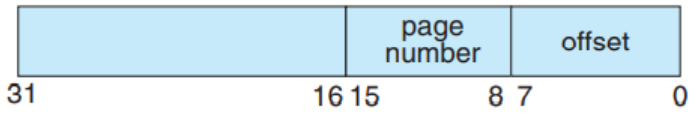
Total Points: 150 Points

Description

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size $2^{16} = 65{,}536$ bytes. Your program will read from a file containing logical addresses and, using a TLB and a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address. Your learning goal is to use simulation to understand the steps involved in translating logical to physical addresses. This will include resolving page faults using demand paging, managing a TLB, and implementing a page-replacement algorithm.

Implementing the Virtual Memory Manager

Your program will read a file containing several 32-bit integer numbers that represent logical addresses. However, you need only be concerned with 16- bit addresses, so you must mask the rightmost 16 bits of each logical address. These 16 bits are divided into (1) an 8-bit page number and (2) an 8-bit page offset. Hence, the addresses are structured as shown as:



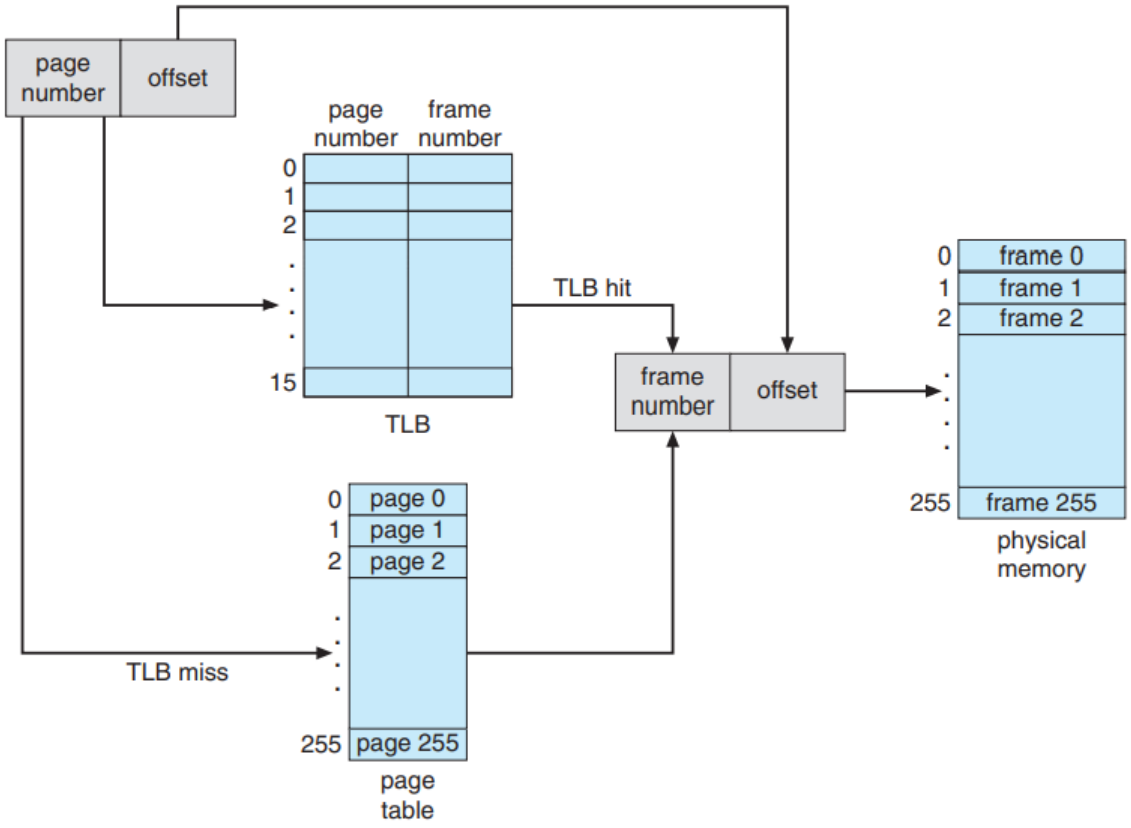
Other specifics include the following:

- 2^8 entries in the page table
- Page size of 2^8 bytes
- 16 entries in the TLB
- Frame size of 2^8 bytes
- 256 frames
- Physical memory of 65,536 bytes (256 frames x 256-byte frame size)

Additionally, your program need only be concerned with reading logical addresses and translating them to their corresponding physical addresses. You do not need to support writing to the logical address space

Address Translation

Your program will translate logical to physical addresses using a TLB and page table as outlined in Section 9.3. First, the page number is extracted from the logical address, and the TLB is consulted. In the case of a TLB hit, the frame number is obtained from the TLB. In the case of a TLB miss, the page table must be consulted. In the latter case, either the frame number is obtained from the page table, or a page fault occurs. A visual representation of the address translation process is:



Handling Page Faults

Your program will implement demand paging as described in Section 10.2. The backing store is represented by the file BACKING STORE.bin, a binary file of size 65,536 bytes. When a page fault occurs, you will read in a 256-byte page from the file BACKING STORE and store it in an available page frame in physical memory. For example, if a logical address with page number 15 resulted in a page fault, your program would read in page 15 from BACKING STORE (remember that pages begin at 0 and are 256 bytes in size) and store it in a page frame in physical memory. Once this frame is stored (and the page table and TLB are updated), subsequent accesses to page 15 will be resolved by either the TLB or the page table.

You will need to treat BACKING STORE.bin as a random-access file so that you can randomly seek to certain positions of the file for reading. We suggest using the standard C library functions for performing I/O, including fopen(), fread(), fseek(), and fclose().

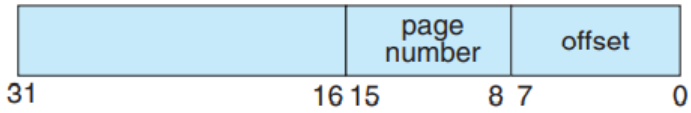
The size of physical memory is the same as the size of the virtual address space—65,536 bytes—so you do not need to be concerned about page replacements during a page fault.

Test File

We provide the file addresses.txt, which contains integer values representing logical addresses ranging from 0to65535 (the size of the virtual address space). Your program will open this file, read each logical address and translate it to its corresponding physical address, and output the value of the signed byte at the physical address.

How To Begin

First, write a simple program that extracts the page number and offset based on:



from the following integer numbers:

1, 256, 32768, 32769, 128, 65534, 33153

Perhaps the easiest way to do this is by using the operators for bit-masking and bit-shifting. Once you can correctly establish the page number and offset from an integer number, you are ready to begin. Initially, it is suggested that you bypass the TLB and use only a page table. You can integrate the TLB once your page table is working properly. Remember, address translation can work without a TLB; the TLB just makes it faster. When you are ready to implement the TLB, recall that it has only sixteen entries, so you will need to use a replacement strategy when you update. **You will use the Least Recently Use replacement strategy or the TLB.**

For each address that is translated you will print the following:

- The logical address being translated (the integer value being read from addresses.txt).
- The corresponding physical address (what your program translates the logical address to).
- The **signed** byte value stored in physical memory at the translated physical address.

Use the following printf statement:

```
printf("Virtual Address: %5d Physical Address: %5d Value: %3d\n", ...);
```

Replace ... with the actual values of the virtual address, the physical address and the value read.

Statistics

After completion, your program is to report the following statistics:

- Page-fault rate—The percentage of address references that resulted in page faults.
- TLB hit rate—The percentage of address references that were resolved in the TLB.

Since the logical addresses in addresses.txt were generated randomly and do not reflect any memory access locality, do not expect to have a high TLB hit rate.

Extra Credit(30 Points) - Page Replacement and Smaller Physical Address Space

This project has assumed that physical memory is the same size as the virtual address space. In practice, physical memory is typically much smaller than a virtual address space. This phase of the project now assumes using a smaller physical address space with 128 page frames rather than 256. This change will require modifying your program so that it keeps track of free page frames as well as implementing a page-replacement policy using LRU (Section 10.4) to resolve page faults when there is no free memory

What to submit

4. Please fill in README.md file given for the assignment
5. Push all completed code to your given repository by the deadline.

How to submit

- git add .
- git commit -m ” message”
- git push