CS446 Assignment 3
Operating Systems, Winter 2022

Question [30 points]: In this assignment you are to implement a Virtual Memory Manager.

You are to write a C program that translates logical addresses 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 Translation Look-aside Buffer (TLB) as well as 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. The goal behind this assignment is to simulate the steps involved in translating logical to physical addresses.

Other specifics include the following:

1. No. of entries in page table = 256 =2^8
2. Page size = 256 = 2^8 bytes
3. No. of entries in the TLB = 16
4. Frame size = 256 = 2^8 bytes
5. Number of frames = 256
6. Physical memory of 65,536 bytes (256 frames × 256-byte frame size)

Your program will need the following header files:

```
#include <stdio.h>
#include <sys/mman.h> /*For mmap() function*/
#include <string.h> /*For memcpy function*/
#include <fcntl.h> /*For file descriptors*/
#include <stdlib.h> /*For file descriptors*/
```

Since the logical address space is of size 2^{16}, the number of bits to represent a logical address is 16. These 16 bits are divided into (1) an 8-bit page number and (2) 8-bit page offset.

There are three parts to this assignment:

Address Translation

Your program will translate logical to physical addresses using a TLB and page table as outlined in the lecture notes under paging.

1. The text file 'addresses.txt' can be downloaded from Canvas. This file contains integer values representing logical addresses ranging from 0 − 65535 (the size of the virtual address space).
2. Your program will take this file as the **first command line argument**.
3. Your program will open this file and read each logical address from the file and compute its page number and offset using bitwise operators in C. See slides on bitwise operators in C under lecture notes on main memory.
4. After extracting the page number from the logical address, look up the TLB. In the case of a TLB-hit (an entry corresponding to a page number exists), the frame number is obtained from the TLB. In the case of a TLB-miss (**NO** entry corresponding to the page number exists), look up the page table. In the latter case, either the frame number is obtained from the page table, or a page fault occurs.
5. Page table can be implemented as an integer array. Initialize all the entries of this table to -1.
6. The below figure gives a nice representation of the address translation process.
7. **Refer to lecture notes on Main Memory and Lab 8 material to work on this part of the assignment.**

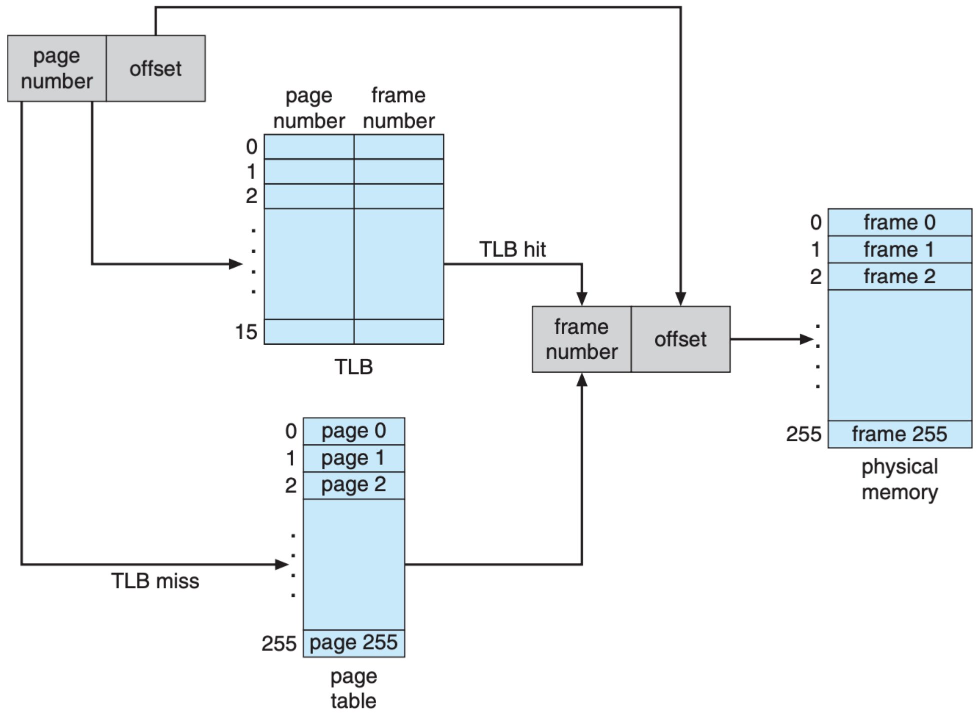

Figure 9.34 A representation of the address-translation process.

Handling Page Faults

1. The backing store is represented by the file BACKING_STORE.bin. This file is a binary file of size 65,536 bytes. Download this file from Canvas.
2. You are to take this file as a **second command line argument**.
3. Open this file using the `open()` system call and map it to a memory region using the system call `mmap()`.
4. When page fault occurs, you will read in a 256-byte page from this memory mapped file and copy it in an available page frame in physical memory using the `memcpy()` function.
5. For this you will need to keep track of available frames. Since virtual memory and physical memory are of the same size, this can be done by simply maintaining a variable that holds the value of the next available frame.
6. Additionally, physical memory can be simulated as a signed character array.
7. The idea is that, if a logical address with page number, for example 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.

8. **Refer to lecture notes on memory mapped files under lecture notes on Virtual Memory and Lab 9 material to work on this part of the assignment.**

Using TLB

1. To simulate a TLB in your program, create a data structure (named as TLBentry for example) to store entries of TLB (page number and frame number pair), and implement it as a circular array consisting of items of type TLBentry.
2. You are to create two functions, one to search the TLB for an entry and the other to add an entry to the TLB.
3. When you add an entry to the TLB, simply use the FIFO replacement policy, which is to replace the oldest entry.
4. It is recommended that you bypass the TLB and use only a page table initially, and once you have your program working correctly integrate the TLB. **Note that, address translation can work without a TLB; the TLB just makes it faster.**

Program Output

Your program is to output the following values:

a. Logical address being translated (that is the integer value being read from addresses.txt).
b. The corresponding physical address.
c. The signed byte value stored at the corresponding physical address.
d. **Compute and output Statistics:** In addition to the above output your program is to report the following statistics:
   i. Total number of page-faults — the number of address references that resulted in page faults.
   ii. Total number of TLB hits — the number of address references that were resolved in the TLB.
   iii. These statistics can be computed by simply maintaining integer variables to keep track of the page faults and TLB hits.

Correct Program output: ./assignment3 addresses.txt BACKING_STORE.bin
****

**** Sample correct output posted on Canvas (correct.txt file).

Deliverables:

- **Assignment3.c - You are to provide a C file named assignment3.c that contains the entire solution for the assignment question.**
- **output.txt**

Assignment 3 Grading Rubric

Assignment 3 is for 30 marks. 10 marks for each section.

30 marks - For the correct solution. Your program implemented all 3 sections of the assignment correctly and as required, and your program output was correct.
25 marks - Program logic is correct for all 3 sections, but your program did not use bitwise operators in C. OR program logic is correct for all 3 sections but outputs incorrect page fault count, and TLB Hit count.
20 marks - Program logic is correct for all 3 sections but outputs incorrect physical address and values in addition to page fault count, and TLB Hit count. OR program logic is correct, and outputs correct values, but your program did not implement TLB.
15 marks - The program implemented all 3 sections; the logic is mostly correct.
10 marks - Program only computed page and offset and did not implement page faults or TLB section.
0 marks - No solution received.