Programming Project 4 – Virtual Memory Manager

Deadline: **Friday April 13 by midnight**
Type: **Group of 2 students maximum**
Weight: **This programming project is worth 5% of your final grade**
Submission: **Must be on Moodle**

Policy for late hand-in:

One-day delay will result in 20% mark reduction. Two days delay will result in 40% mark reduction. After that, the assignment will not be accepted.

Marking Scheme:

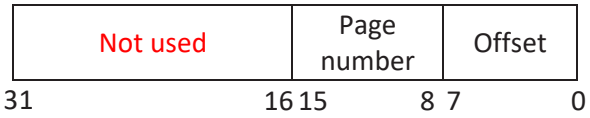
- Code correctness (75%)
- Code output format, clarity, completeness, and accuracy (10%)
- Report (15%)

Goal

This programming project involves implementing and simulating a virtual memory manager, a program that translates logical to physical addresses for a virtual address space of size 65,536 bytes. The program will read from a file containing several logical addresses. Using a TLB (translation look-aside buffer) as well as a page table, it will then translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address. Essentially, the program will simulate the steps involved in translating logical to physical addresses.

Specification

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 two parts: 1) an 8-bit page number; and 2) 8-bit page offset. Thus, the addresses are structured as shown in the figure below.



Other characteristics of the virtual memory manager include the following:

- 256 entries in the page table.
- Page size of 256 bytes.
- 16 entries in the TLB.
- Frame size of 256 bytes.
- 256 frames.
- Physical memory of 65,536 bytes (256 frames $\times$ 256-byte frame size).

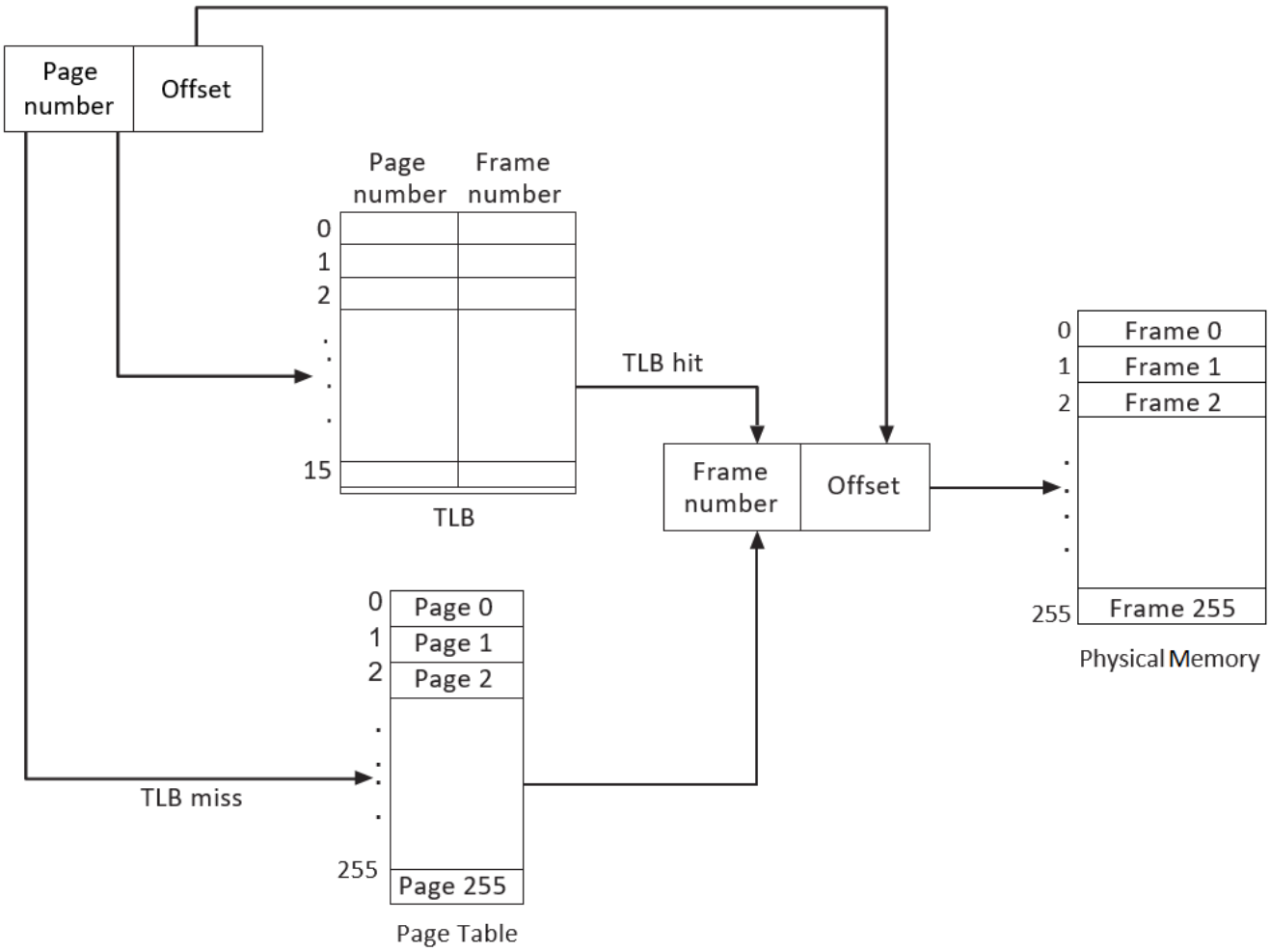
Part I

In this part, your program need only be concerned with reading logical addresses and translating them to their corresponding physical addresses. It does 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 explained in the course slides. 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 appears in the following figure.



Handling Page Faults

Your program will implement demand paging. 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** 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()`.

In this part of the project, the size of physical memory is the same as the size of the virtual address space (65,536 bytes). Thus, you do not need to be concerned about page replacements during a page fault. In part II, you will use a smaller amount of physical memory. At that point, a page-replacement strategy will be required.

Test File

The file addresses.txt is provided, which contains integer values representing logical addresses ranging from 0 to 65535 (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 the figure provided above) from the following integer numbers:

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

Perhaps the easiest way to do this is by using 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, we suggest that you bypass the TLB and use only a page table. You can integrate the TLB once your page table is working properly. Remember that 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 16 entries, so you will need to use a replacement strategy when you update a full TLB. You may use either a FIFO or an LRU policy for updating your TLB.

How to Run Your Program

Your program should run as follows:

```
./a.out addresses.txt
```

Your program will read in the file addresses.txt, which contains 1,000 logical addresses ranging from 0 to 65535. It translates each logical address to a physical address and determines the contents of the signed byte stored at the correct physical address. (Recall that in the C language, the `char` data type occupies a byte of storage, so we suggest using character values). Your program should then output the following values:

1) The logical address being translated (the integer value being read from the file addresses.txt).
2) The corresponding physical address (what your program translates the logical address to).
3) The signed byte value stored at the translated physical address.

The file correct_output.txt is also provided, which contains the correct output values for the file addresses.txt. You should use this file to determine if your program is correctly translating logical to physical addresses.

Statistics to be Output

After completion, your program should report the following statistics:

1) Page-fault rate: the percentage of address references that resulted in page faults.
2) 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.

Part II

Part I assumes that physical memory has the same size as the virtual address space. In practice, physical memory is typically much smaller than a virtual address space. A suggested modification is to use a smaller physical address space. You should use 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 either FIFO or LRU. Your program should output the same values and statistics as in part I.

Companion Material

Three files are provided:

- addresses.txt
- correct.txt
- BACKING_STORE.bin

Deliverable

The deliverable consists of the following:

1) A copy of a well-commented source code and a sample run (pbs_output.txt).
2) A three pages report specifying a high-level description of the program. The report should also include a brief conclusion.
3) In your report, indicate the status of your program by specifying whether:
   - The program runs with user-defined test cases.
   - The program runs with some errors.
   - The program compiles and runs with no output.
   - The program does not compile.