# Lab #6 – Virtual Memory

Due: Thursday, April 28 (beginning of class)

This lab 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 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 of this lab is to simulate the steps involved in translating logical to physical addresses.

## Specification

Your code will read a file containing several 16-bit integer numbers that represent logical addresses. These 16 bits are divided into (1) an 8-bit page number and (2) 8-bit page offset. The addresses are structured as shown in Figure 1.

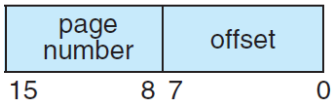


Figure 1. Address structure

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 × 256-byte frame size)

Additionally, your program need only be concerned with reading logical addresses and translating them to their corresponding physical addresses.

## Address Translation

Your code will translate logical to physical addresses using TLB and page table. 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. The address translation process is shown in Figure 2.

## Logical Addresses

You need to use the file `addresses.txt`, which contains integer values representing logical addresses ranging from 0 – 65535 (the size of the virtual address space). Your code 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.

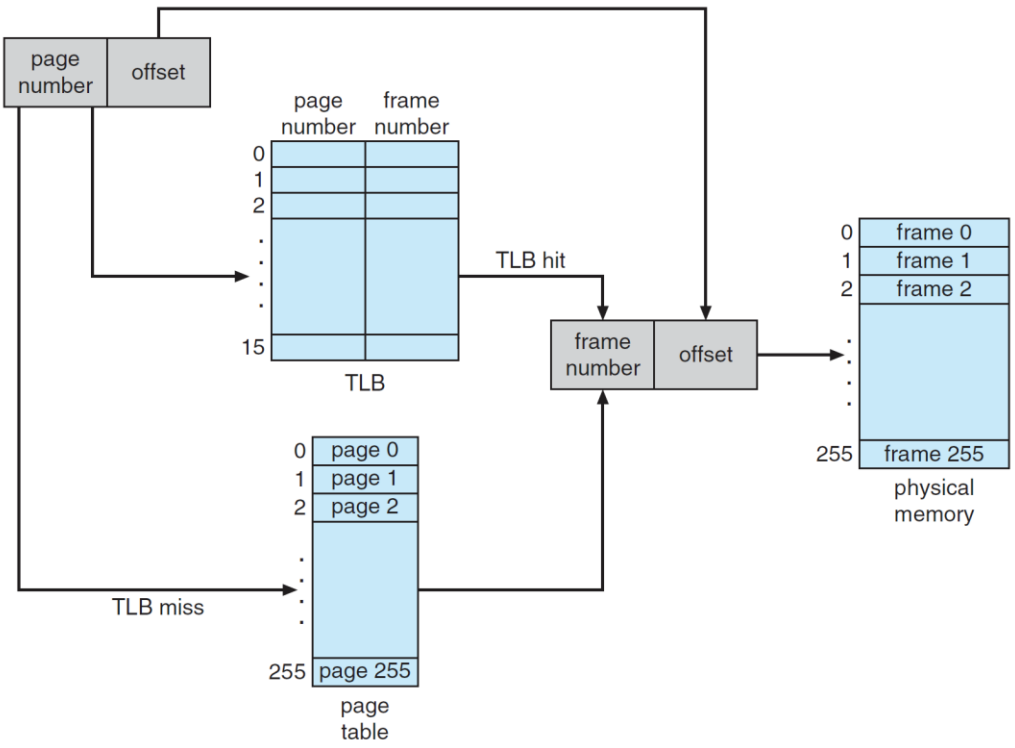


Figure 2. Address-translation process

**Handling Page Faults**

Your code 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 code 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. You are suggested to use 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.

**Step I**

First, write a simple code that extracts the page number and offset (based on Figure 1) from the following integer numbers:

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

The easiest way to do this is by using the operator 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, you can 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 16 entries, so you will need to use a replacement strategy when you update a full TLB. You may use a FIFO policy to update your TLB.

**Step II**

Your code should run as follows:

```
./virtmem  BACKING_STORE.bin  addresses.txt
```

Your code will read in the file `addresses.txt`, which contains 1,000 logical addresses ranging from 0 to 65535. Your code is to translate each logical address to a physical address and determine 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 you need to use `char` values.)

Your output consists of the following values:

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

To create a new mapping in the virtual address space, you may need to use `mmap()` function. To copy page from backing file into physical memory, you may need to use `memcpy()` function. In addition, you may need the following:

```
#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

#define TLB_SIZE 16
#define PAGES 256
#define PAGE_MASK 255

#define PAGE_SIZE 256
#define OFFSET_BITS 8
#define OFFSET_MASK 255

// Max number of characters per line of input file to read.
#define BUFFER_SIZE 10
```

**Result**

After completion, your code 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.

For example, your output will be shown below:

```
Virtual address: 16916 Physical address: 20 Value: 0
Virtual address: 62493 Physical address: 285 Value: 0
Virtual address: 30198 Physical address: 758 Value: 29
Virtual address: 53683 Physical address: 947 Value: 108
Virtual address: 40185 Physical address: 1273 Value: 0
Virtual address: 28781 Physical address: 1389 Value: 0
Virtual address: 24462 Physical address: 1678 Value: 23
Virtual address: 48399 Physical address: 1807 Value: 67
Virtual address: 64815 Physical address: 2095 Value: 75
.
.
.
.

Virtual address: 48065 Physical address: 25793 Value: 0
Virtual address: 6957 Physical address: 26413 Value: 0
Virtual address: 2301 Physical address: 35325 Value: 0
Virtual address: 7736 Physical address: 57912 Value: 0
Virtual address: 31260 Physical address: 23324 Value: 0
Virtual address: 17071 Physical address: 175 Value: -85
Virtual address: 8940 Physical address: 46572 Value: 0
Virtual address: 9929 Physical address: 44745 Value: 0
Virtual address: 45563 Physical address: 46075 Value: 126
Virtual address: 12107 Physical address: 2635 Value: -46
Number of Translated Addresses = 1000
Page Faults = 244
Page Fault Rate = 0.244
TLB Hits = 54
TLB Hit Rate = 0.054
```

**Submission:**

Make the followings as `cmpe320lab6_lastname.tar.gz` and then upload it on Dropbox.

- Makefile
- Readme
- All source codes
- Any extra files needed to run your program
- Documentation
    - Description of the program
    - Algorithm
    - Output

***You need to submit the hardcopy of your documentation and all source code on due date.*** Your file should include student, course, and instructor information. Here is an example.

```
/*
* Lab 6: Virtual Memory
* Programmer: your name and your partner's name
* Course: CMPE 320
* Section: 1 (9-10:50am) or 2 (11-12:50pm)
* Instructor: S. Lee
*/
```

**Marking Criteria:**

- Submission of required file only, with the information of student, instructor, and course on the submitted file
- Warning free compilation and linking of executable with proper name
- Readability, suitability & maintainability of source code and makefile
- Documentation
- Demonstration