

# Project 4: Address Translation

This is to be an individual effort. No partners.

For this assignment, you will **create the file p4.c**, which implements virtual to physical memory mapping using bit-level operators for the given scheme:

- **Virtual address** – 17 bits total with a 10-bit offset. This means we are using 7 bits to hold the VPN, so the page table will have up to  $2^7$  entries. The TLB is a direct mapped cache using 3 index bits and 4 tag bits in the same manner as described in class.
- **Physical address** – 25 bits total – the PPN will be 15 bits with the 10-bit offset.

Note that other than the number of bits used overall, this is similar to the setup in the memory examples from class.

Both the virtual address and physical address are fewer than 32 bits in length and will fit into a standard 32 bit integer. **You must use an integer (and bit-level operators) to do your work.** Use masks and shifting to get to the parts of the address you need to use at a given time.

## Notes:

- your own program, **p4.c**, expects a command-line argument naming the page table (without .pt extension), which it should open and read, storing internally as you see fit. Then your program enters a loop, expecting virtual addresses (in decimal), and attempting the translation to physical address via the TLB or page table.
- The TLB is represented in your program internally (using an int for each entry) however you see fit; it begins empty, and updates its entries as PT hits occur.
- you must include and use the provided **log.c** and **log.h** files' provided logging functionality. Call **start\_logging()** at the beginning of your program; call **log\_it()** each time a lookup attempt has been performed to record the outcome; and call **stop\_logging()** at the end (to save the file). Be sure to update the user id in the **start\_logging()** function!
- the provided **translate** executable can be used (on **zeus**) both to test program functionality and to auto-generate page tables (by feeding a command-line argument of the desired page table name):

```
msnyde14@zeus-1:~$ ./translate example
Enter a virtual address or -1 to exit: 123
Virtual Address: 0x7b
Physical Address: 0x188a87b from the page table
Enter a virtual address or -1 to exit: 123
Virtual Address: 0x7b
Physical Address: 0x188a87b from the tlb
Enter a virtual address or -1 to exit: 1100
Virtual Address: 0x44c
Page fault
Enter a virtual address or -1 to exit: -1
demo$ ls
example.pt ...
```

- though you will want to print messages out directly for the user interactions, be sure you also call `log_it()` to record the translation outcomes, as this is where grading will focus. There are four kinds of logging actions; look at the `log.c` file for plenty of guidance on how to call `log_it()`.
  - ILLEGAL:** when the address is out of range. (remember, we're not simulating a 32-bit or 64-bit system, so there are too-large addresses possible!)
  - FROM\_PT:** lookup was successful, but had to use the page table to find it.
  - FROM\_TLB:** lookup was successful, and the TLB had the entry cached.
  - PAGE\_FAULT:** there was no valid entry in the PT for this address.
- Your program will be given the page table (as a command-line argument) but you will implement and maintain a TLB for each run of your program. The TLB is initially empty – when you get a page table hit, you will update the TLB appropriately so that the next time an address with the same physical page number is referenced, you will be able to get it from the TLB instead of the page table.

Another example of what a run of your program might look like is given below. The underlined text is the user input, and some interesting parts are color-coded.

```
demo$ translate snyder
Enter a virtual address or -1 to exit: 1234
Virtual Address: 0x4d2
Page fault
Enter a virtual address or -1 to exit: 12345
Virtual Address: 0x3039
Physical Address: 0xa52c39 from the page table
Enter a virtual address or -1 to exit: 12346
Virtual Address: 0x303a
Physical Address: 0xa52c3a from the tlb
Enter a virtual address or -1 to exit: 1234567
Illegal virtual address
Enter a virtual address or -1 to exit: -1
demo$
```

## Implementation Notes

- You must use an integer (and bit-level operators) to do your work.** Use masks and shifting to get to the parts of the address you need to use at a given time.
- Your program must read the page table from the input file. The root of the name of the input files is given on the command line. For example, `p4 test` will indicate the file `test.pt` should be opened. The file structure is pretty simple and described below. (you can also automatically generate them with our provided `translate` tool).
- Page table file format – because we have 7 VPN bits, we have  $2^7 = 128$  lines, each with a valid bit (0 or 1) and a physical page number (in decimal). This physical page number will be no more than 15 bits. An example of this is below.

```
1 4645
0 2184
1 13765
1 22917
1 3066
1 13038
0 8032
...
```

## Submitting & Grading

Submit this assignment on blackboard as a tar file. Be sure to include all needed files:

- a valid **Makefile** that builds your program on zeus,
- **p4.c**
- **log.c** and **log.h** files (remember to update the user name)
- any others needed to build and run your assignment.

Your grade will be determined as follows:

- 80 points – Correctness.
- 20 points – use of C, comments, bit-level operators, etc.