Operating Systems
CSCI-UA.0202 Fall 2013

Programming Assignment 2
The TLB

Due Monday, November 25

Note: This is not a group project. Each student must write his or her own code and not use the code from other students.

This assignment is to implement a TLB (translation lookaside buffer) within a simulated memory management system. The details concerning the TLB are described in the files `tlb.h` and `tlb.c`. You are welcome to use the code that is already in `tlb.c`, but you will need to fill in the rest of the needed code.

The TLB should be implemented as follows:

- It should be represented as an array, where each element of the array (i.e. each entry of the TLB) is contains a valid bit, a virtual page number, an M bit, an R bit, and a page frame number.
- Upon every instruction generated by the (simulated) CPU, the MMU will call `tlb_lookup(vpage, op)`, which – if there is a TLB entry for the virtual page `vpage` – should set the R bit, set the M bit (if the `op` is a `STORE`), and return the page frame. If there is no entry for `vpage`, the variable `tlb_miss` should be set to TRUE.
- After a TLB miss, when the MMU, using the page table, has determined the page frame, it will call `tlb_insert(new_vpage, new_pframe, new_mbit, new_rbit)`. This should cause the specified new virtual page, page frame, M bit, and R bit values to be written to an entry of the TLB. Which entry is chosen is determined by the clock algorithm described in `tlb.c`. If a valid TLB entry has to be evicted, the M and R bits for the evicted page frame should be written back to the corresponding bits of the M and R bitmaps maintained by the MMU (see the functions `mmu_modify_rbit_bitmap()` and `mmu_modify_mbit_bitmap()` in mmu.h).
- When a page fault occurs, the MMU will call `tlb_write_back()`, which should write the value of the M and R bits of each entry in the TLB to the M and R bitmaps in the MMU. This is to allow the M and R bitmaps to be used in selecting a page to evict from memory to make room for an incoming page.
- After a page fault, the OS will call `tlb_insert()`, as above, once the desired page (which caused the page fault) has been loaded into a page frame in memory.
- On a clock interrupt, the OS's clock interrupt handler will call `tlb_clear_all_R_bits()`, which clear the R bit in every TLB entry.

What you need to do

You will need to have the gcc C compiler installed on your computer, as for the previous project.

The first thing you need to do for this assignment is download and unpack, using the same method as for the previous assignment, the gzipped file for the machine you are using. Follow the link on the course web page corresponding to the operating system (Windows, Mac OS X, or Linux) that your computer runs.

I have implemented the following components: the CPU (compiled to `cpu.o`), the MMU (`mmu.o`), the page table (`page.o`), and the OS memory management code (`kernel.o`). Your only task is to implement the TLB in the file `tlb.c`. When you are finished, email me just your `tlb.c` file.

What you are provided with

In addition to the compiled files `cpu.o`, `mmu.o`, `page.o`, and `kernel.o`, you are also provided with the header files, `cpu.h`, `mmu.h`, `page.h`, `kernel.h, and types.h`. Furthermore, you are provided with `tlb.h` and some skeleton code in `tlb.c`, which you can choose to use or not as you wish.

To compile the entire program using your `tlb.c`, type

```
gcc –m32 –o proj2 tlb.c cpu.o mmu.o page.o kernel.o
```

Alternatively, as with the first assignment, I have provided you a Makefile, so all you need to do is type

```
make proj2
```

The executable file generated by the compiler will be named `proj2.exe` (using cygwin on a PC) or just `proj2` (on a Mac or Linux machine). In either case, to run the program just type

```
./proj2
```

I have provided a compiled version of the entire program in the file `ben` (or `ben.exe`, depending on the machine you are using). You can compare the output of my program to the output of yours, to see if you are on the right track.

When you run the program there are several options you can provide, as follows:

`-v` verbose: print extensive message describing the events occurring while the simulation is running

`-t`*N* set the number of TLB entries. The number of entries must be a power of 2; the program automatically selects the smallest power of 2 no less than N.

`-p`*N* set the maximum number of virtual pages in the simulated process (also a power of 2)

`-f`*N* set the number of page frames in memory (a power of 2).

`-n`*N* set the number of memory instructions (LOADS and STORES) issued by the CPU to *N*.

For example,

```
./proj2 –v –t100 –f1000 –p10000 –n100000
```

will turn on the verbose feature, set the number of TLB entries to 128 (the smallest power of 2 no less than 100), set the number of page frames to 1024, set the number of virtual pages to 16384, and the number of instructions issued to 100,000.

Try running `ben` with these various options to see what the output looks like. If no options are given, the simulation will use default values for each of these options, and the verbose feature will be turned off.

Using the `-v` option causes the variable `verbose`, declared in `cpu.h`, to be set to true. In your TLB code, you should use this variable to determine whether to print information about the events that are occurring in the TLB. For example, in `tlb_insert()`, if a TLB entry is being evicted, you might put the statement:

```
if(verbose)
    printf("Evicting TLB entry for pageframe %x ", ... );
```

Let me know if you have any questions.