CPSC 213: Assignment 10

Due: Wednesday, April 4, 2012 at 6pm.

Late assignments are accepted until Friday, April 6 at 6pm with a 25% penalty per day (or fraction of a day) past the due date. This rule is strictly applied and there are no exceptions.

Goal

In this assignment you extend the Simple Machine simulator to implement page-based virtual memory and then observe a test program running with two separate page tables.

Adding Virtual Memory to the Simple Machine

You will use your implementations of `MainMemory` and `CPU` from previous assignments and a new class called `VirtualMemoryCPU`. A skeleton of this class is provided for you: add the file `VirtualMemoryCPU.java` to your Eclipse project in the same directory as the other two files.

You will implement the method `translateAddress` of this class, which translates a virtual address to a physical address. This method needs to read from memory using a physical address in order to read the page table. To read from memory using a physical address use `physMem` instead of the `mem` used in CPU to read memory. Both `mem` and `physMem` implement the same set of methods; the only difference is that `mem` uses virtual addresses and `physMem` uses physical. The other thing you will need to do is to read the value of the page table base register, the base physical address of the current page table. This register, which is settable in the SimpleMachine GUI (bottom left), is read by the statement `ptbr.get()`.

To run the virtual-memory version of the simulator you use a different target (or different command-line option). The new target is the class `SimpleMachine.Sm213VmStudent`. If you select the machine using command-line arguments, change "`-a sm213`" to "`-a sm213-vm`".

Requirements

Here are the requirements for this week's assignment.

1. Implement the `translateAddress` method of `VirtualMemoryCPU` and test your implementation. Use a page size of 32 bytes, as already specified in the template code (much smaller than the 4K page size example presented in lecture).
2. Edit the `min.s` program you wrote for assignment 4 to use your implementation of virtual memory, as follows. First, change the locations of your code segments to be the smallest possible addresses. Next, create a second copy of the data section at a different position in memory than the first one; again, use addresses that are as low as possible.

Next, create two additional data sections in which you will manually make two page tables. Each page table should map a different data section, but they should both use the same code (which you should not have duplicated).

3. Run your version of `min.s` twice, once using each page table so that each execution uses a different set of inputs. Look closely at what happens as the program runs. Record your observations.

Notes

1. A page table is just an array of `.long` values. It is an array indexed by virtual page numbers, where the values within it are physical page numbers. That is, when you access a virtual address that falls within the *i*th chunk of addresses, the function `translateAddress` will use the value of the *i*th entry in the page table array to calculate the physical address.
   Consider the page size to figure out how large of a chunk of addresses each entry should handle. Unused virtual pages should be marked as invalid. The table only needs to be large enough to contain the last valid page-table entry. You do not need to worry about virtual addresses beyond the end of the page table in this assignment.
   You may find it helpful to put a comment after each line documenting both its index (the vpn), and the virtual address that you intend as the result after the physical address value is translated.
2. Be careful in your assembly language program with assigning labels. The simulator has a limitation that can cause confusion. Labels must refer to virtual addresses, because that is what the CPU uses when it executes your code. However, the addresses you specify with .pos in the .s file are PHYSICAL addresses. This approach is a programming expediency as virtual memory is a recent add-on to the simulator.
   So, if you put a label on a line of code or a data element whose virtual address is different from the physical address specified in the `.s` file your label will have the wrong virtual address when the program runs. For example, if you copy the data section of your program and also copy the labels, then the second copy of `min`, `i` and `a` will have the wrong virtual address and your program will not run.
   The solution is to ensure that the code and the first copy of the data have virtual addresses that are the same as their physical addresses, and to only put labels in the code and the first copy of the data. The second copy of the data can be at any physical address --- just don't put any labels next to these `.long` values.
   Alternatively, you can avoid using labels entirely, and use only addresses in your code.
3. Remember that all addresses used within the assembly code are virtual addresses: for example, `ld (r0), r0` would take the address stored in `r0`, interpret it as a virtual address, translate the virtual address into a physical address, and the value stored at that physical address would be loaded into `r0`. The translation that happens is dependent on the page table mappings that you specify. Remember that all `.pos` directives in your assembly file are physical addresses. Remember that page tables are indexed by vpn and contain pfns as values.
4. The page table entry has two fields encoded in different bits of a single integer. The top bit is the valid bit; everything else is the PFN. The two masks that you need to easily

obtain the desired bits are already created for you in the template file: PTE_PFN_MASK and PTE_VALID_BIT_MASK.

5. For testing, you will set the current page table by setting the `ptbr` to store the appropriate base address for each page table in turn. Do this by using the simulator GUI (bottom left corner) to manually set the `ptbr`, then doubleclick on the first instruction to set the PC there (turning it green), and hit Run. To run the second time, manually set the `ptbr` again to the address of the other page table, click the first instruction again, and then hit Run again.

Material Provided

The template for `VirtualMemoryCPU.java` is in the file `code.zip`.

What to Hand In

Use the **handin** program. The assignment directory is **a10**. Please hand in exactly the following files with the specified names. Do **NOT** hand in class files, or your entire Eclipse project, or a README in formats like `.doc` or `.rtf`.

1. `VirtualMemoryCPU.java` with implementation of `translateAddress` method, as specified in Requirement 1. Your test code may be either included in this file, or in a separate file that you should also include.

2. Your modified `min.s` assembly file, as specified in Requirement 2.

3. `README.txt` that contains:

   - header with your name, student number, four-digit cs-department undergraduate id (e.g., the one that's something like a0b1)

   - statement that "I have read and complied with the collaboration policies" at http://www.ugrad.cs.ubc.ca/~cs213/winter11t2/policies.html

   - description of your testing of `translateAddress`, as specified in Requirement 1.

   - description of your observations from running the modified `min.s`, as specified in Requirement 3