Cmpt 250 Assignment #4 12-2

Revised: **2012-07-26**
Due Date: 11:59pm, Friday, Aug. 3, 2012
Late Policy: 10% off per calendar day

There are two major sections in this assignment.

- Analyzing and working with a DesignWorks computer similar to that in Assignment #2, except it has cache.
- Doing some basic cache and virtual memory (to be covered in class soon) calculations that are similar to what might be asked on the final exam.

You may do these two sections in any order.

In the first section, you are going to analyze a DesignWorks computer that is very similar to the solution to Assignment #2. However, your TA has done a great job of employing buses to clean up the design. He has written his own Mux and own simplified general purpose register that can be cleared and loaded. Finally, he has added a cache between the PC and the ROM. However, we have removed 3 essential lines of VHDL in the cache component. Those lines update the correct line cache when there is a cache miss.

You are first going to study the VHDL code to figure out what the cache-augmented controller's state machine looks like (which adds a small time delay as necessary when there is a cache miss). You MUST comment the code as you go, and hand the commented VHDL in. As described below, you'll also document the state machine of the controller with an ASM. Once you understand the code, you will then add the missing three lines of VHDL.

Part A

1. Download the file 250a4.12-2.starterFiles.zip from the Assignment portion of the course web site.
2. The above file runs only on DesignWorks, so you'll have to do this assignment using the CSIL assignment lab, or remote desktop from home into leto.cs.sfu.ca. When you get the .zip file to CSIL, you MUST place it on your H:\ drive. Uncompress it to some location on your H\: drive. Do not put it or the uncompressed files on your "desktop".
3. Examine the MySimpleCPU.cct to get a sense of how the whole computer works (very similar to Assignment #2). Note if you double click a VHDL component, or try to simulate this circuit, you will likely have to accept some dialog windows that pop up asking you if your project should attach to the .dwv files in the directory that you have placed it in (in contrast to where we had it when we zipped it up to send to you). You should be fine if you accept each attachment.
4. Open the controller.dwv file. Add VHDL comment documentation to make the working of the state machine very clear (you will later need to hand in the commented VHDL). In particular, add to everything related to the S3 state. As you are going along, start sketching an ASM style diagram for the state machine. Here are several hints for the state machine:
   - Draw this in landscape orientation (so the paper is wider than tall).
   - The behavior with a cache hit, and behavior after the state that handles a cache miss, are extremely similar. You should be able to draw a line from state S3 to just after the S0 rectangle to avoid a huge amount of duplication in your diagram.
   - To do an ASM branch on the opcode, use an extremely-wide hexagon. See slide 6-51 and label the 4 outputs of the hexagon with the 4 different opcodes.
   - In the ASM, please only show register transfers into the Acc and the PC. You also only need to specify the control assertions that are "true" for each rectangle or oval (you can just write their name).

- Please add miscellaneous comments to your diagram regarding which instruction is being executed, and whether a branch is taken or not.

Submit your final ASM diagram and your more commented controller.dwv file.

5. Make a small table of how many clock ticks each instruction should take. Have three columns: Instruction, Ticks with Miss, Ticks with Hit. Include this table in your submission.
6. Analyze the VHDL code in the cache.dwv component. Instead of a single wide table, the cache is implemented in 3 arrays. Think of each as a column of the cache table. There is an array for the valid bit column, an array for the tag column (tag_bank), and an array for the data column (data_bank).
   The style of this VHDL component is to define the types, the cache storage, and miscellaneous internal vector signals that are used at architecture scope. Most of the work in this program happens outside the single process of the entity; it takes place in the architectural scope signal assignments. You must follow *how* each architecture scope signal assignment thence affects the right side of another architecture scope signal assignment. Please add VHDL comments to each type and signal definition, and explain what each architecture scope signal assignment is doing. Note how some of the assignments are extracting important details from the input address, some are assigning to other useful internal signals, and others are using internal signals to assign results to output ports. While looking at the code, try to answer the question below.
7. Question: Even 3 missing lines of code missing, you should be able to answer whether the code implements direct-mapping, N-way mapping, or fully associative mapping? (Hints: Do you see a wide data array (or multiple data arrays) for a given tag? Do you see code that does a broad search of all the tags for a given incoming tag value?)
8. In the cache component, after the VHDL text "if (cache_hit = '0') then", please add three missing lines of code to update the three fields of the cache line selected to be overwritten. Comment your code lines. (Hint: This is not hard, though the left hand sides of the assignments are more complicated than the right hand sides).
9. Run the project, slow the simulation down, and for several seconds put the reset switch to '1'. You've seen this ROM program before: load 0, increment 7 times, add 8, load 0, and branch back if zero. But watch the timing diagram; if the cache was helping, most instructions would not take three clock ticks.
   Question: Is the problem in your code, or is this program just not very well suited to the size and type of cache?
10. Change the ROM program to something that will allow some successful cache hits. You might even be able to make a program that after a very short boot-up has every fetch result in a cache hit.
11. Record a timing diagram of some instructions executing and each not starting with a miss. Save it or paste it into your primary submission document.

Part A Submission Requirements:

- A4: Submit your final ASM diagram, and your more commented controller.dwv.
  Note: We would appreciate you submitting your whole working Designworks project. It should be in a directory called Assign-4, and you can just include your assignment primary submission document (.docx or .pdf) in that same directory, then zip it up.
- A5: Table of instruction durations with miss and with hit.
- A6 and A8: Submit your revised cache.dwv file. Be sure it has extensive comments.
- A7: Answer the question in step A7.
- A9: Answer the question in step A9.
- A10: Submit your ROM_16x6.dwv file containing your modified program.
- A11: Submit a timing diagram showing at least one instruction that doesn't begin with a miss.

Part B

For the next several parts, I'm going to ask you a number of questions about cache that could be on the final exam.

1. Question: Is the above a data cache, instruction cache, or both?
2. Question: What is the cache width (measured in bits) for the above cache?
3. Question: What is width of the cache block index in the above design?
4. Question: How many lines does the above cache have?
5. Question: What is the tag width? Knowing this width and the index width, how many blocks does it signify the actual memory (not cache) has?

Part B Submission Requirements (in this order)**:**

- Submit the answers to questions B1 – B5 in your primary assignment document.

Part C

Assume a new CPU has a 64KB memory with a direct-mapped cache. Words are 2-bytes wide. Cache blocks are just one word.

1. Assume you have 64 cache lines. What is the width of the cache line index?
2. What is the width of the byte index?
3. Draw a diagram similar to that on slide 13-57, and below it label the bit boundaries very well. Within each cell you can name the field and indicate its bit width.
4. What is the width of the tag?
5. CPU manufacturers like to brag about the sizes of their caches measured in bytes. What is the size of this cache in bytes (not including the valid bit)?

Part C Submission Requirements:

- Answer questions C1 through C5 in your primary submission document.

Part D

1. See the diagram on slide 13-75 for a direct-mapped cache. What is the maximum amount of physical memory that the processor could have?
2. How many different tags are there for the whole of physical memory? How many tags can fit in the cache (this is super simple)? So how many blocks have to share the same cache line?
3. If you doubled the width on slide 13-75 of the top box denoting the breakdown in bits of an address, how many cache lines would there need to be? It's not required, but we strongly recommend that you first do a logical address breakdown diagram similar to that in slide 13-57, and additionally put the cell name and width within each cell of the diagram.
4. What size of resulting multiplexer would be needed? How many selector inputs would that multiplexer need?
5. What would be the resulting width of a word measured in bytes?
6. What would be the resulting size of the cache itself measured in words and also measured in bytes (not including the "valid" bit nor the "tag" column)?
7. What is the resulting width of the tag? So if your processor had enough RAM for the entire address space, how many cache blocks in memory have to share the same cache line in this resultant direct mapped cache (you can express your answer as a power of two)?

Part D Submission Requirements:

- Answer questions D1 – D7 in your primary submission document.

Part E

This part concerns the topic of virtual memory. We may not get entirely finished this topic during the July 24 lecture, but we will finish it on the July 31 lecture.

1. Consider the diagrams on slide 14-24 and slide 14-26. For the physical address size at the bottom of slide 14-26, what is the maximum number of bytes in physical memory (this first question doesn't really concern VM).
2. How many bits wide does the page directory table need to be ~~if it selects one of 1024 page tables~~?
3. If the page directory table also needs 4 bits for "valid", "dirty", "referenced" and "read-only" bits, how wide does a page directory table need to be?
4. If the above width must be rounded up to a whole number of 8-bit bytes, how many bytes wide does the page directory table need to be?
5. How many bytes does an entire page directory table, so rounded up, require?
6. For the same slides 14-24 and 14-26, how wide does the address coming out of the page table have to be?
7. Given that you might need the same 4 special additional bits, how many bytes wide does the page table need to be. How many bytes does a whole page table of this width need to be?
8. How many page tables might you need in principle for this virtual memory system?
9. If all the page tables are needed, how much total space do all the page tables take up in total (excluding the page directory table)?

Part E Submission Requirements:

- Answer questions E1 – E7 in your primary submission document.

Submission:

- Please review the document in the Assignment section of the course web site titled "DocAndSubmissionRequirements.pdf".
- I have listed each submission requirement of this particular assignment at the end of each Part of this assignment; check to be sure you have included them all.
- The primary submission document (MS-Word or .pdf) MUST have a title page containing near the very top the course, the semester, the assignment number, your lastname, firstname, your student ID, and your email address.
- Your primary submission document must have a table of contents on the title page.
- You will submit your primary document and (if required) a number of Designworks/Logicworks files electronically.
- Put all your files in a directory called Assign-N where N is the assignment number.
- You'll have to have the ability put this whole directory in one .zip file and submit just that one container file. You can use WinZip, WinRAR, or other zip utility (most of these are free for non-commercial use). Winrar is already installed on leto.csil.sfu.ca.
- Submit electronically to our CourseSys at http://courses.cs.sfu.ca/ in a .zip called Assign-N.zip where N is the assignment number. The files should be in a subdirectory of the .zip called Assign-N.