Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

ECE 312

Computer Organization and Design

Spring 2001

MP2: The LC-2 Processor with a Unified 4-Way Set-Associative Cache PART II

Version 1.0

J. Strasser
S. J. Patel

TABLE OF CONTENTS

Chapter 1: Introduction

Checkpoint 1 of MP2 left you with a working machine that implements the complete LC-2 Instruction Set. Now that we have a machine that we can fully program and operate, we can start to look at performance. Memory caches are vital for achieving high performance in microprocessors. In lecture we have talked about why caches help processor performance and some of the key characteristics of different cache designs. To help you gain a more detailed understanding of caches, how they work and their performance, you will be augmenting the LC-2 design from checkpoint 1 with a simple one-level cache. Specifications of the cache are in chapter 2.

Again, this handout is only intended to help you get started. You are responsible for implementing the additions and modifications to the MP2 Checkpoint 1 design on your own. We will cover cache design in lecture. In addition, your textbook may have helpful information on caches. We have put together some information to help you get started (see the Getting Started Chapter). Only start this checkpoint when you have a correct implementation of Checkpoint1. Refer to the LC-2 appendix for ISA concerns.

This final checkpoint will require the correct implementation of the LC-2 instruction set with a unified set-associative cache and is **due Friday, March 9th at 5pm**. The final turn-in is worth 85% of the MP2 grade. The specifics of the required cache configuration are covered in another chapter. **You are strongly encouraged to write your own test code to test your implementation and not wait until we release ours. This includes verifying not only instruction implementation, but cache correctness.**

Grades will be based on the correctness of design, style of design, thoroughness of verification, and quality of documentation. Specifics of what to turn in are covered in chapter 6. MPs can be turned in up to two days late with an automatic 20 point deduction (out of 100). MPs will not be accepted after two days past the due date.

The remainder of this MP2 write-up contains helpful reference information for completing the MP.

Chapter 2: Cache Specifications

You will need to design a one level, unified, 4-way set-associative cache. Each Data Store should have sixteen lines with 1 word/line. You will need to create four tag store arrays, four data store arrays, and four valid bit arrays. Refer to chapter 5 for memory array delays. These arrays can do no computation, just decoding the address and performing the functions of a read/write to a complete line.

The replacement policy of this cache is the pseudo-LRU described in H&P page 606. In addition to the arrays above, you will need to create one more vhdl array for the LRU bits. You may not handle LRU computation within the LRU memory array.

These thirteen memory array boxes should contain vhdl to decode an address and do an appropriate read or write. You are allowed to use a for loop to initialize LRU and Valid bits in your cache design. There should be only five ways that a memory array interfaces with other components: an address bus, two data busses (data in and data out), RESET_L and a write signal.

Your cache will be a write-through cache with a write no-allocate policy. You will need to copy your memory.vhd to a new memory block within your memory design.

You may not add additional signals to memory.vhd. Your cache must work with the same signals that communicated with memory in Checkpoint 1. The datapath should have no knowledge of your memory hierarchy. Signals used are found in chapter 3.

For this MP you are not allowed to model the cache as a simple renoir vhdl block (i.e. making a single block and then writing straight vhdl to model the cache behaviorally). You must construct the cache only using the following components: **registers, control unit (create a state diagram for this), decoders, comparators, muxes and logic gates** *(see design constraints for delays associated with these components)*. For LRU store, valid store, the tag store and data you may use renoir blocks to model these array structures. VHDL code for main memory has been provided (it is the same code as in Checkpoint 1 in which main memory access time has been set to 500ns for a word access).

Chapter 3: Signal Specifications

Memory Subsystem Signals

ADDRESS(15:0)
: Memory is accessed using this 16-bit signal. It specifies the address that is to be read or written.

DATAIN(15:0)
: 16-bit data bus receiving data from memory.

DATAOUT(15:0)
: 16-bit data bus sending data to memory.

MREAD_L
: Active low signal that tells memory that the address is valid and that the processor is trying to perform a memory read.

MWRITE_L
: Active low signal that tells memory that the address is valid and that the processor is trying to perform a memory write.

MRESP_H
: Active high signal generated by memory indicating that the memory has finished the requested operation.

The convention that is used for all signal names (except data paths, e.g. ADDRESS, DATA, etc.) is to append an underscore and polarity to the end of the signal name. For example, XYZ_H indicates that the signal XYZ is active when high.

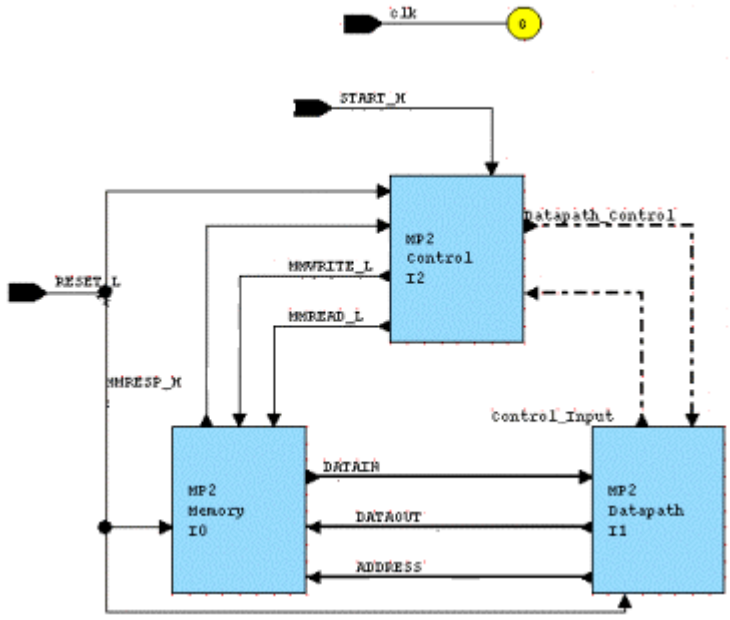
Bus Control Logic

The memory system is asynchronous, meaning that the processor waits for the memory to respond to a request before completing the access cycle. In order to meet this constraint, inputs to the memory subsystem should be held constant until the memory subsystem responds. In addition, outputs from the memory subsystem should be latched if necessary.

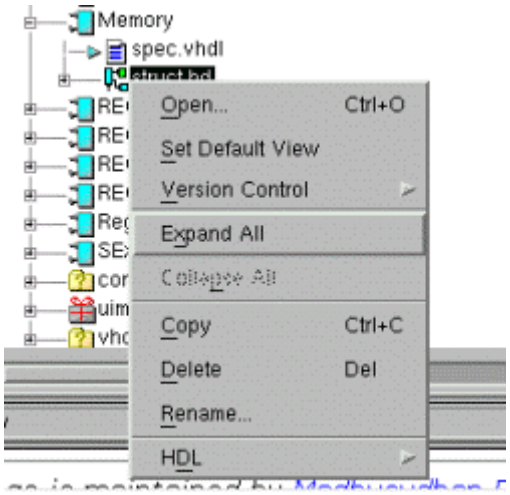
The processor sets the MREAD_L control signal active (low) when it needs to read data from the memory. The processor sets the MWRITE_L signal active when it is writing to the memory. The memory activates the MRESP_H signal when it has completed the read or write request. We assume the memory response will always occur so the processor never has an infinite wait.

Chapter 4: Getting Started

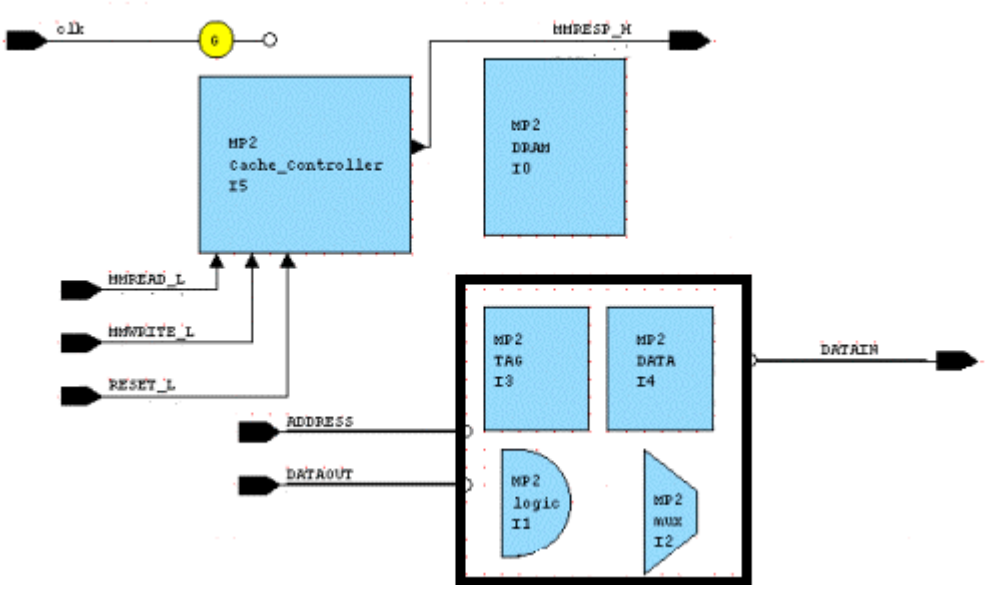
To design the new memory hierarchy, follow these steps:



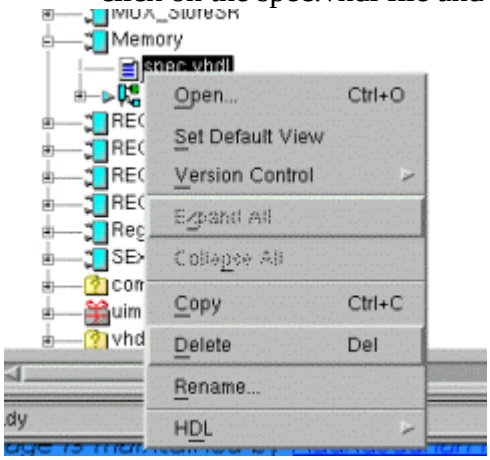
1) First click on the Memory block using the right mouse button
2) A menu will open up. Select **Open-> New View**
3) Make sure that under **Select Type of View to Create,** "Block Diagram" is selected. Click **OK**. A new block diagram window will appear.
4) Switch to the Design Browser window. Locate the Memory block and click on the small plus to the left of it. You should see a **spec.vhdl** and a **struct.bd**. Right click on the struct.bd file and then click on the **Set Default View** option.



5) Double click on the **struct.bd** icon. You can now begin drawing your design in this window as the example below demonstrates. The Cache_Controller block is a Renoir stage diagram. *Note: The following diagram is NOT complete nor necessarily correct!*



6) Now you have two files associated with your memory block(struct.bd and spec.vhdl) but you only need one. Switch back to the Design Browser window so that you can delete the spec.vhdl file. Find the memory block in Design Browser, right click on the spec.vhdl file and then select delete.



7) You will have a main memory block(labeled DRAM in the above example) and vhdl code this has been provided to you. You will want to cut and paste the contents of the memory vhdl into your new main memory block(the block that accessed when a miss occurs in the cache). To paste in the new memory VHDL code, first open up the code in your web browser or text editor. Then double click on your

memory block, an option to create a new diagram will appear, choose vhdl, name the file if you'd like. A text editor window will open up and you can paste the new code over the default vhdl that appears in the window. Save it and then close the window

8) You are now ready to begin your cache design. Good luck!

Chapter 5: Design Limitations for Checkpoint 2

As with checkpoint 1:
You Cannot:

Add additional ports in/out to the LC-2 register file
Design non-"realistic" VHDL. For example, muxes should be nothing but a switch statement

You Can:

Make additional muxes with the following limitations:
1. The delay for a 4 input MUX is 2 * that of a 2 input MUX.
2. The delay for an 8 input MUX (including 5,6,and 7 input variations) is 4 * that of a 2 input MUX

Add logic gates with the following limitations:
1. Logic is limited to 4 inputs per gate
2. The delay for a 2-input gate is 1 ns.
3. The delay for 3 and 4 input gates is 2 ns.

Add additional port in/outs between cpu components (datapath and control).

Lengthen your clock to as much as 50 ns to accommodate your critical path. Make sure that the MRESP_H signal is held high for the length of your clock cycle. This signal is found in the Memory.vhdl file.

You may use the **to_integer** function in you memory arrays, do not use a decoder outside of the array.
Sext has the delay of a two-input mux (sext should use the msb to select between 0's and 1's).
Zext has no delay (it's just a concatenation).
Concatenation requires no delay (bunching or splitting wires incurs no delay).
Delays need to be coded into your individual component files.

Component Delays
The following components can be used in your cache design.

Comparator delay = 5ns
Memory array delay = 20ns (this includes tag, data, LRU and valid arrays)

Chapter 6: MP2 Checkpoint 2 Hand-in Requirements

Checkpoint 2 is due Friday, March 9th by 5pm, despite what the timeline says.

Please turn in the following in this order:

1. All Datapath diagrams
2. Control Unit diagrams including all hierarchical diagrams
3. Simulation trace of the last 100 CYCLES, ending when the last instruction of the test code has finished execution. In the trace please include:

   CLK, RESET_L, START_H, ADDRESS, DATAIN, DATAOUT, Opcode, Out_ALU, Out_PC, MREAD_L, MRESP_H, RegWrite, Dest, Reg(0) through Reg(7), all tag, data, valid, and LRU storage (just add them as you would any signal by finding their instances).
   All values should be displayed in HEX. Please highlight your final register values.
4. Please include a trace showing a cache hit and highlight your hit signal.
5. Memory Dump with all store instruction values.

Use the test code provided from the webpage (to be supplied closer to the due date).
You are strongly encouraged to write your own test code to test your implementation and not wait until we release ours.

To perform a memory dump, please do the following:

1) Enter ModelSim
2) Open the Lists Window
3) Drag signals MWRITE_L, DATAOUT, and ADDRESS into the Lists Window

Next, you need to set the triggering for each of the three signals. Select a signal in the list window and go to Props->Signal in the menu. The triggering option should be at the bottom right of the options window. Set the triggering as follows:

MWRITE_L - Triggered
DATAOUT - NOT Triggered
ADDRESS - NOT Triggered

Now just run the simulation and every memory write should be recorded in the lists window. Write the output to a *.lst file, print, and you're done.