ECE 411: Computer Organization and Design

MP 1: The LC-3b Processor

Version 4.0

Table of Contents

1. Introduction

Congratulations on completing MP 0. You now have the design of a preliminary LC-3b processor. However, it only implements 6 of the LC-3b instructions. In MP 1, **you're going to finish what you started in MP 0 by implementing the rest of the LC-3b instructions**. The full LC-3b ISA can be found on the course website under the "lab" menu (http://courses.engr.illinois.edu/ece411/mp/LC3b_ISA.pdf). In the next MP, you will add a cache to your processor. When you are done with both MP 1 and MP 2, you'll have a fully working LC-3b processor with a cache!

Unlike MP 0 which contained step-by-step instructions, this handout is only intended to help you get started. You are responsible for implementing the additions and modifications to the MP 0 design on your own. Datapath and control design was discussed in ECE 385 and in MP 0. In addition, Section 4.4 of Patterson and Hennessey (4th ed.) provides an example of another design, which might supply you with ideas on how to proceed.

The rest of this document gives instructions and describes expectations for this MP. Make sure that you read section 3 and 4 before starting. Grades will be based on the correctness of design, style of design, thoroughness of verification, and quality of documentation.

2. Important Resources

This section is meant to prevent and alleviate problems you don't want to encounter as the MP progresses.

2.1 Website

MP updates, important announcements, files to help you test and debug, and hand-in information will be posted on the course website. You are responsible to stay up-to-date with announcements posted on the website, so be sure to check it regularly!

https://courses.engr.illinois.edu/ece411/mp/faq.php is a good source for frequently asked questions from the previous semesters.

2.2 Piazza

Piazza is the best resource for getting help and tips. The TAs monitor this resource regularly and often fellow students will answer your question. **Be sure that you have read all previous posts before posting your question; your question may have been addressed already**.

Many problems and questions that the TAs encounter during office hours are already addressed, discussed, and answered on Piazza. You will save yourself a lot of time and trouble by regularly checking Piazza. By learning from other students' problems, you can save yourself from making the same mistakes. Also, tips, updates, and clarifications from TAs will help make your MP experience more pleasant. Note that **not all material on Piazza will be cross-posted to the website**, so it is important to check Piazza regularly for updates from the course staff.

3. The LC-3b Instruction Set Architecture

For this project, you will implement the rest of the LC-3b instruction set architecture (except for the RTI instruction). For a complete description of the LC-3b ISA, refer to the ISA reference manual available on the Lab portion of the course web page (http://courses.engr.illinois.edu/ece411/mp/LC3b_ISA.pdf). As you will reference this document throughout the rest of the semester, we highly recommend you print this document and keep it on you when you are working on your MP.

All sixteen instructions are 16 bits in length, having a format where bits [15:12] contain the opcode. The LC-3b ISA is a load-store ISA, meaning data values must be brought into the general purpose register file before they can be operated upon. Each general purpose register is 16 bits in length (meaning that the LC-3b is a 16-bit ISA). The address space of the LC-3b is accessed using 16 bits (meaning that the LC-3b memory consists of 2^{16} locations). Each location contains a single byte (meaning that the LC-3b memory is byte-addressable). The general purpose register file contains 8 registers.

As in MP 0, the LC-3b program control is maintained by the Program Counter (PC). The PC is a 16-bit register that contains the address of the next instruction to be fetched. Please note that we have provided you with example test code for the AND (immediate), ADD (immediate), TRAP, LDI, STI, LDB, and STB instructions in the Machine Problem FAQs on the course website. Of course, you should also write your own test code—successfully running the provided test code does not guarantee the correctness of your design. For every instruction, you should carefully verify with a test that the instruction implementation indeed updates the state registers that it is supposed to change and as importantly, it does not change the state registers that it is not supposed change.

4. The LC-3b Memory Control

4.1 Memory Control Signals

Your microprocessor design will use the same signals to communicate with the outside world as the processor of MP 0 did. This includes the signals `mem_write` and `mem_byte_enable` which control writes to the memory. Descriptions of these signals are provided below.

`mem_write`
: Active high signal that indicates that the address is valid and that the processor is performing a memory write.

`mem_byte_enable`
: A mask that indicates which byte(s) should be written. The behavior of this signal i indicated in the following table.

MSB	LSB	Behavior
0	0	Write to neither byte of a word when mem_write becomes active
0	1	Write only to the low byte of a word when mem_write is active
1	0	Write only to the high byte of a word when mem_write is active
1	1	Write to both bytes of a word when mem_write is active

4.2 Memory Control Logic

The bus control logic is similar to that of MP 0. The processor sets the appropriate mask in `mem_byte_enable` and asserts `mem_write` active when it is writing to memory (note: you did not need to work with the masks in MP0, in this MP, you will need to use them properly for implementing some instructions in the ISA). Assume that the memory response will always occur so that the processor never has an infinite wait.

5. Getting Started

Since MP 1 is an extension to the work done in MP 0, we will require students to copy their completed MP 0 design into a new folder for MP 1.

5.1 Create a Copy of the MP 0 Design

1. Ensure that you make a backup of your MP 0 design before attempting the conversion. This will help prevent potential deletion disasters before they occur. **For this and future MPs, it is highly recommended that you periodically backup your projects**. To create a backup, navigate to your project and create a compressed tarball.

```
$ cd ~/ece411
$ tar -cJvf mp0_backup.tar.xz mp0/
```

2. Copy your MP 0 project directory into a new directory which you will use for MP 1.

```
$ cp -r mp0 mp1
```

3. In the new directory, change all references to mp0 to reference mp1 instead (e.g., mp1_tb instead of mp0_tb, mp1.sv instead of mp0.sv). You can do this manually or try to use the *rereference.sh* script

```
$ cd mp1/
$ ../bin/rereference.sh . mp0 mp1
```

4. Download the memory.sv file from https://courses.engr.illinois.edu/ece411/mp/mp1/memory.sv and put it into your MP 1 directory. Go to *Assignments* -> *Settings* -> *Simulation* to modify the testbench. Just like MP 0, click the *Test Benches…* button, click to highlight your existing test bench (refer to MP 0 document to create a new one if it is not there), and click *edit* button. In the *Edit Test Bench Settings* window, remove the *magic_memory.sv* and add *memory.sv*, and click OK. After finishing configuring the Settings, open *mp1_tb.sv,* and change the instantiation of *magic_memory* to *memory.* The new memory module describes a memory that does not respond immediately like the magic memory. Using this memory, you will need to make sure that you are handling the memory response signal appropriately.

5. Modify the `DEFAULT_TARGET` variable in the *bin/load_memory.sh* script so that the memory initialization file is written to the MP 1 simulation directory.

```
# Settings
DEFAULT_TARGET=$HOME/ece411/mp1/simulation/modelsim/memory.lst
ASSEMBLER=$HOME/ece411/bin/LC3bAssembler
ADDRESSABILITY=1
```

5.2 Preliminary Test

After you have copied over your MP 0, you should compile and simulate the design to make sure that the copy-over procedure was successful. Do not continue with MP 1 until you have verified that the copied MP 0 design still works. If it does not work, make sure you followed the steps above and/or consult a TA for help.

5.3 Paper Design

Once you have verified that your MP 0 was copied over correctly, begin the design of your complete LC-3b datapath **on paper**. You can use the MP 0 datapath diagrams provided in the *mp0/spec/* directory as a starting point for your MP 1 design. For each new instruction, trace its execution through the existing datapath and decide what components need to be added or modified to allow said instruction to be processed.

5.4 Design Input

Once you have completed the paper design for an instruction (or for all instructions), implement the modifications in Quartus and test your design.

6. Design Limitations

6.1 Things You Must Not Do:

- Begin MP 1 when your MP 0 does not work
- Add any additional ports to the LC-3b register file
- Add additional datapath registers (on top of IR, PC, MDR, MAR)
- Add additional inputs to the ALU
- Use the same output from the IR for all the immediate values. Imm5, TrapVect8, Index6, Offset9, and more are provided for you in the types package
- Add logic in-between the MDR and mem_wdata bus
- Add logic (besides the mdrmux) in between the mem_rdata bus and MDR.
- Modify the memory

6.2 Things You May Do:

- Add additional muxes and widen existing muxes
- Add logic gates

7. Checkpoint Handin

To help keep you on pace with this MP, we will have a checkpoint before the final handin is due. For this checkpoint, you must have the following added to your MP 1 design:

- ADD immediate instruction
- AND immediate instruction
- JUMP/RET instruction
- LEA instruction

Note: for the final handin, you will be implementing all of the LC-3b instructions, not just the ones listed above.

Test code is provided at https://courses.engr.illinois.edu/ece411/mp/mp1/mp1-cp.asm to verify the correct functionality of these instructions in addition to the ADD, AND, NOT, LDR, STR, and BR instructions you have already implemented in MP 0.

For the checkpoint handin, submit a wave trace showing the **last 1000ns** of test code execution to the ECE 411 dropbox. The wave trace must include the below signals in the order presented:

1. `clk`,
2. `pc_out`,
3. `mem_address`,
4. `mem_read`,
5. `mem_rdata`,
6. `mem_write`,
7. `mem_byte_enable`,
8. `mem_wdata`,
9. registers from register file (expanded)

Highlight the final value of each register in the register file.

Rename the signals to remove the prefixes. For example: remove "/mp1_tb/dut/mp1_datapath/...etc…" and leave only the actual signal name as listed above.

8. Final Handin

8.1 Implementation

For the final handin, you should have implemented **all** of the LC-3b instructions (with the exception of RTI). For a list of instructions see the LC-3b ISA document.

8.2 Getting the test code:

The final test code will be uploaded on Thursday before the deadline. Once the test code is released, it can be found at: http://courses.engr.illinois.edu/ece411/mp/mp1/mp1-final.asm. It tests all the instructions. However, passing these test codes doesn't necessarily mean that your design is working in all cases. You need to write your own test code to cover more corner cases.

8.3 What to hand in:

You need to hand in the following things. Please read the list carefully! Failure to follow all instructions could cause you to lose points.

1. A **paper design of your Datapath** block diagram.
   - You may use the design from MP 0 as a starting point for your MP 1 design. Files containing the MP 0 design can be found in the *mp0/spec/* directory.
   - **Highlight** all blocks which you have modified from MP 0. ("Modified" means that you changed the SystemVerilog code inside the block, or replaced the block entirely)
   - **Highlight** all blocks you have added.
   - Please keep this paper design to a minimum number of pages while still maintaining legibility. "Legible" means features and text are visible to the unstrained naked eye.
   - Your paper design should be detailed enough for the TAs to trace the execution of LC-3b instructions through your processor. The level of detail should be comparable to the datapath design given for MP 0.
   - You may wish to keep a copy of your paper design for your own reference.

2. **Descriptions of highlighted blocks**.

For each highlighted block (except highlighted muxes) on your datapath diagram, give a short description of its functionality. These descriptions may not be written on the datapath diagram. Make sure that each description includes (at least):

- A description of what the block does.
- What instruction(s) use the block.

Some made-up examples:

- "MagicSuperBlock takes its inputs, concatenates them, and shifts left by one bit. This is done to support the XOR instruction's special addressing mode."
- "SuperDuperBlock rips bit 7 off its input. This value is output as HappySignal. HappySignal enables the SHF instruction by determining which direction to shift."

3. An **explanation of how you tested** your design.
Write several paragraphs explaining your strategy for testing your design. We want to know how you tested your design, NOT how you designed it. Also include an explanation of why you believed that your design would run the test code properly. Note that just saying that you ran the provided test code successfully does not constitute a proper explanation. To substantiate your testing strategy, you should submit at least one test code that you wrote while doing your debugging and explain how it tests your design. You will be graded on your description of your testing strategy. Do NOT hand-in the code we have provided you.

4. A **wave trace of the first 2000ns** of test-code execution.
Please hand in a one page printout of the wave trace of the final test code we have provided you with on the course website. The wave trace must include these signals (**in order**):

1. `clk,`
2. `pc_out,`
3. `mem_address,`
4. `mem_read,`
5. `mem_rdata,`
6. `mem_write,`
7. `mem_byte_enable,`
8. `mem_wdata`
9. registers from register file (expanded)

NOTE: To print waveform in white background, click the wave window and choose **File -> Print Postscript…**, then choose the range (e.g. From 0 ns to 1000 ns) according to the submission requirement and modify the file name (.ps). Print the created **.ps file**. Do NOT take the screenshot of the waveform which will make the fonts not so legible as well as leave the background blank wasting ink.

Warning: Be sure that the wave traces actually have values on them, not just signal names. If you add signals to a wave after simulating, you must re-run the simulation in order for the new values to be displayed.

5. A **wave trace of the last 1000ns** of the test code.

Again, this is a single page printout of the end of the test code we have provided you. You should show enough time to prove that the program is in the FINISH loop. (If the PC shows the same value several times in a row, this would prove you're in a loop.) On this printout, ***highlight* the final value of each register**. The wave trace must include the same signals as above 4 (first 2000 ns wave trace). Also, all other procedures are identical to 2 (initial wave trace).

6. A **list of all memory writes** from test-code execution.
Creating lists was covered in MP 0. Your list must contain only the following signals (in order):

1. `mem_address`
2. `mem_write`
3. `mem_byte_enable`
4. `mem_wdata`

Set the properties of the `mem_wdata`, `mem_byte_enable`, and `mem_address` signals so that they are displayed in **Hex** and **do not trigger lines**. To do this:

1. Select a signal name in the List window.
2. Select **View -> Properties...**
3. Select "**Hexadecimal**" for the Radix, and "**Does not trigger line**" for Trigger.
4. Click OK.

When you then run your simulation, there should appear two list entries for each memory write (store). If this is not the case, please double-check your settings.The list can be printed by selecting **File -> Export -> Tabular list…** (or in list window, **File -> Write List(tabular)...**). The file you create can then be printed. The file you create can then be printed. This list should fit on one page. Extreme length indicates a problem!

7. A **printout of your timing analysis report** containing the sections listed below (same as the ones submitted for MP 0). You may print only the requested sections. If you submit the entire report, you must **highlight** the requested sections.

The timing analysis report should contain the following sections:

- TimeQuest Timing Analyzer Summary
- Clocks
- Slow 1100mV 85C Model Fmax Summary
- TimeQuest Timing Analyzer Messages (make sure that no messages are being suppressed)

8. We will be using Box for code hand in this semester. Zip your code using

```
$ zip -r <netid>_MP1.zip ece411/mp1
```

And upload using the following url:

https://courses.engr.illinois.edu/ece411/mp/mp1/mp1_upload.html

The prompt will confirm that your code was uploaded successfully to the Box account.

General notes:

- All values on your printouts must be in **HEX**!!
- If you show more signals than those that are required, you should have a good reason for doing so.
- Do **NOT** hand in a wave trace of the entire test code execution.
- Turn the completed MP to the ECE 411 dropbox (ECEB third floor, just north of the student lounge, box #8). The box is labeled.
- Remove signal prefixes (such as "/mp1_tb/dut/mp1_datapath/...etc…") and leave only the signal name as listed in part 4.

9. Grading Rubric

Total points: 100 points

Checkpoint

- **Last 1000 ns Wave: 12 points**

Final Hand-in

- **Design: 24 points**
    - **Paper Design: 8 points**
    - **Block Description: 8 points**
    - **Testing Strategy: 8 points**
- **Functionality: 44 points**
    - **First 2000 ns Wave: 4 points**
    - **Last 1000 ns Wave: 20 points**
    - **Memory Write List: 16 points**
    - **Timing Analysis Report: 4 points**
- **Box Code Hand-in: 20 points**

Appendix. FAQ

For FAQs related to this and other MPs, see
https://courses.engr.illinois.edu/ece411/mp/faq.php#MP1