University of South Florida

CDA 4205L Computer Architecture Lab

Lab Report

Semester: Fall 2023		
Experiment	*Number:*	#8
	Date:	10/18/2023
Lab	*Section:*	Sec003
	Lab TA:	Gil Olenscki Neto
Report	*Due Date:*	10/25/2023
Group ***(Lab3 onwards)***	*Member #1 name:*	Aidan Khalil
	Member #2 name:	Sean Finch

RUBRIC SUMMARY

[10%] For attendance
[90%] Report

ABSTRACT:

[10%] State the purpose and objective of the lab. Give a brief introduction and summary of what you did, what you learned and your conclusion (1 paragraph).

The purpose of this lab was to implement a single-cycle CPU in Verilog HDL and to create the missing instruction memory component. We successfully designed and instantiated the instruction memory, initialized it with machine code instructions, and synthesized the CPU design using Vivado. Through this process, we learned how to declare and initialize memory in Verilog, synthesized the CPU design, and explored its critical path and resource utilization. This lab allowed us to learn more about computer architecture by exploring the inner workings of a CPU and its hardware design. In conclusion, we successfully added the instruction memory to our single-cycle CPU, taking a crucial step towards creating a functional processor for running RISC-V instructions.

INTRODUCTION:

[5%] Write an introduction making sure to explain the lab/experiment (1-2 paragraphs).

In this lab, we strengthened our skills with digital computer architecture as we worked on the creation of a single-cycle CPU using Verilog HDL. The central focus of this experiment is the development of a fundamental component of a fully functional processor: the instruction memory. Building on the foundation from our previous labs, where we had to implement the ALU and Control Unit, our current objective is to construct a comprehensive single-cycle RISC-V CPU capable of executing machine code instructions. However, before reaching that milestone, we must first address a critical missing piece – the instruction memory.

To do this lab we wrote our Verilog code in the instruction memory file and ran the synthesis. Our primary task revolves around integrating the instruction memory module into our CPU design. Through Verilog HDL, we will declare and initialize this memory, populating it with the necessary machine code instructions. As we navigate the hardware considerations involved in CPU implementation, this experiment provides us with hands-on experience in memory design, synthesis, and critical path analysis.

METHOD:

Software Tools Used:

[5%] List all the software tools you need to implement the lab.

- Vivado: FPGA design and synthesis tool
- Verilog HDL: hardware description language used for specifying the behavior & structure of digital circuits.

Files:

[5%] List all files included in your submission. Include a brief description (1 sentence) of each script. Only include files you/your group wrote or modified.

- instr_mem.v: Verilog HDL script used to implement the instruction memory module in our single-cycle CPU design.
- CDA4205L Lab8.pdf: pdf report for this lab.

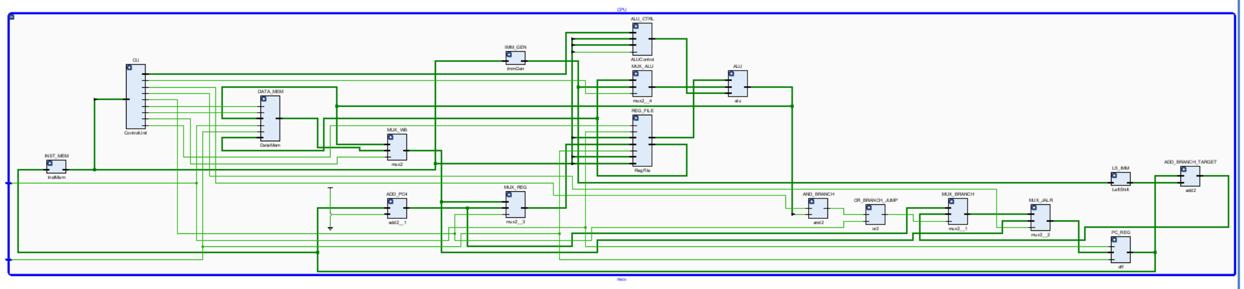
RESULTS:

[40%] Describe your observations of your completed experiment. Include screenshots, plots, tables, or answers to questions, as required, for each lab task (T). Describe any issues experienced in the implementation. If your measured results differ from the expected results, mention how far off you were and provide an explanation. This should be 1 paragraph per task (T). Your answers must be concise, coherent, complete, and correct.

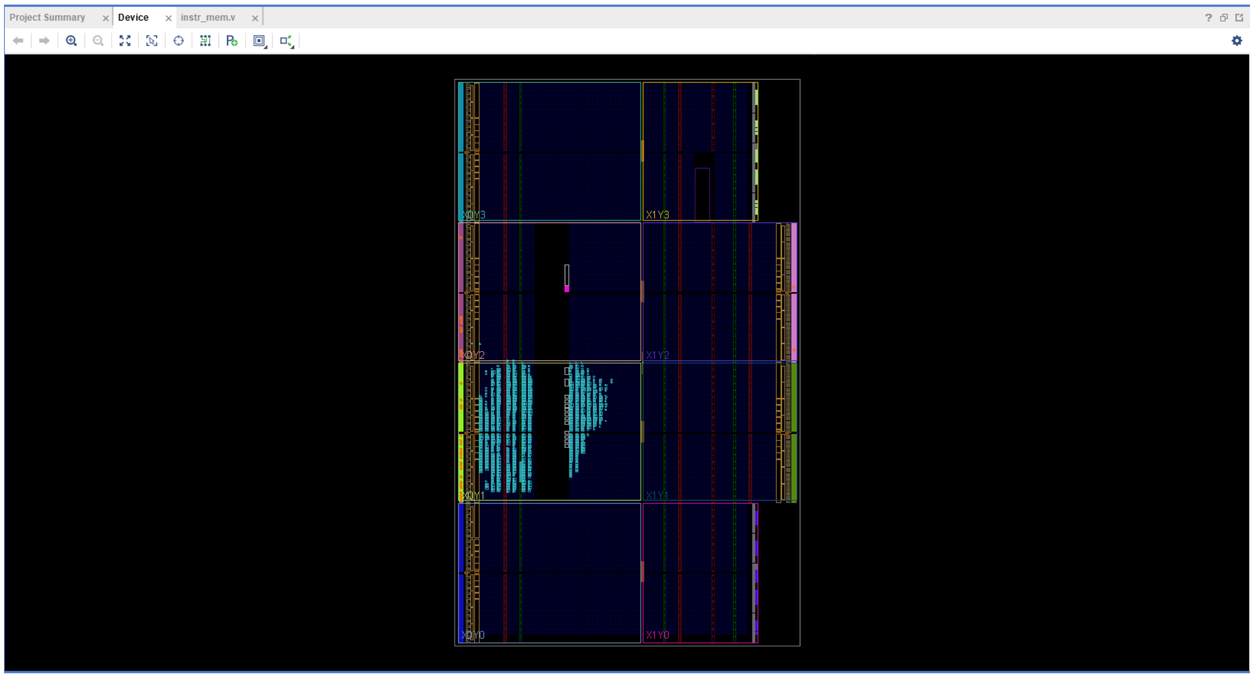
T1: Why do we not need the lower two bits of the address when reading from the instruction memory?

In RISC-V, the instruction memory is byte-addressable, meaning that you can access individual bytes of instruction memory. However, instructions in RISC-V are 32 bits (4 bytes) long and are naturally aligned to 32-bit boundaries. This alignment simplifies the fetching and decoding of instructions, and it's the reason why the lower two bits of the address are not needed when reading from instruction memory.

T2: Take a screenshot of the block diagram generated by Vivado. Remember to click into the CPU block before taking the screenshot so it shows the actual design.



T3: Take a screenshot of the FPGA view in the Device tab.



T4: Report the utilization of logic resources (LUTs, FFs, DSPs) used by the design.

LUTs	FFs	DSPs
2605	1056	3

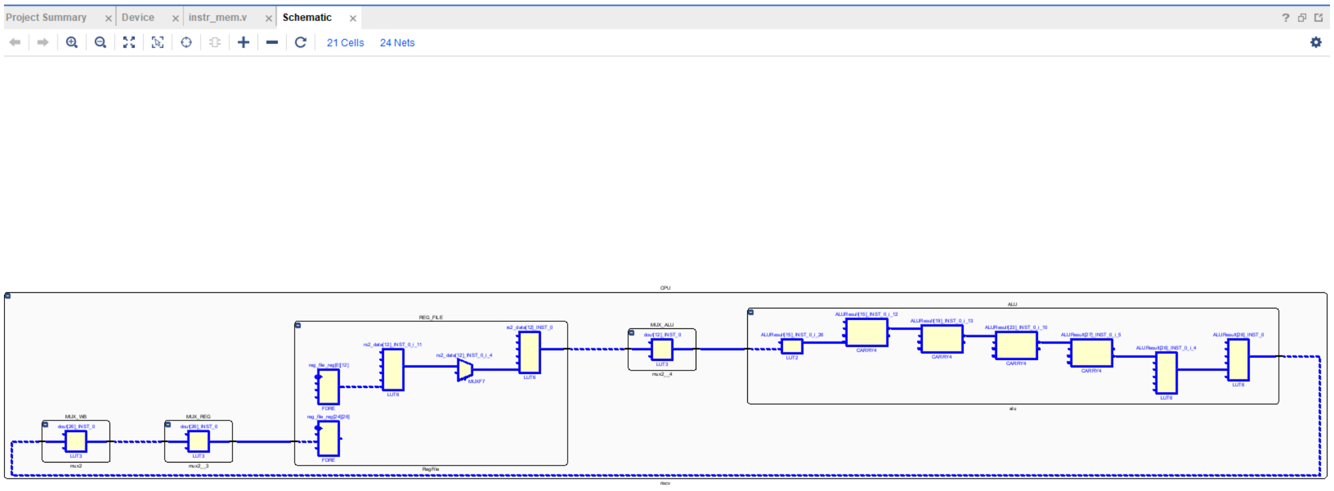
T5: What is the clock period/frequency for this design according to the create_clock command?

The clock frequency is 100 MHz, and the period is 10 ns.

T6: What is the length (in nanoseconds) of the critical path? This will be the path with the highest Total Delay. Based on the clock period, how much "slack" is there in the timing? (That is, how much time is "leftover".)

Insert critical path length (highest total delay): 9.411 ns
Insert timing slack (based on clock period): 0.494 ns seems in the timing.

T7: Take a screenshot of the schematic showing the critical path. What instruction(s) would you expect to use this path?



Through the critical path as illustrated on the schematic, data travels through LUTs, FDREs, MUX, and CARRY4 cells
Based on this knowledge, the expected path instructions include R-type for arithmetic instructions, because our final output returns back into the input we assume no S-type is used. Due to the noticeable lack of immediates or memory accesses, we also assume no I-type instructions were used.

T8: The given implementation only supports full 32-bit instructions, but the RISC-V spec calls for "compressed instructions" as well. Compressed instructions provide a different encoding that stores common instructions in 16 bits instead of 32. How would our InstMem module change if we wanted to support compressed instructions?

To support compressed instructions (16-bit) instead of 32-bit instructions in the InstMem module we'd do the following:
1. Change DATA_WIDTH to 16 bits.
2. Update NUM_MEM_CELLS accordingly.
3. Modify memory initialization to load 16-bit instructions.
4. Update the assign statement to access 16-bit instructions based on the address.

T9: Why is a continuous assignment statement used for reading from the instruction memory? How would this differ if we were implementing a pipelined datapath?

This memory is part of a single-cycle implementation, data is not read synchronously. Hence, we do not use a clock; instead, it is a continuous assignment statement. In simpler terms, the continuous assignment is used for reading from the instruction memory in a single-cycle CPU because instructions are fetched and executed within a single clock cycle without the need for clock synchronization. In a pipelined datapath however, synchronous clock signals are used to control the flow of instructions between pipeline stages, replacing the continuous assignment used in the single-cycle design.

DISCUSSION:

[15%] Discuss the implications of your work and results. Propose potential future work and any shortcomings of your design (2 paragraphs).

Our work in implementing the instruction memory module in the single-cycle CPU design represents a significant step toward creating a fully functional processor capable of running RISC-V instructions. This lab allowed us to gain insights into memory design in Verilog HDL, synthesis using Vivado, and the examination of a CPU's critical path / resource utilization. The successful integration of the instruction memory is important here as it forms the foundation for executing machine code instructions. This work lays the groundwork for further enhancements and refinement of the CPU design, bringing us closer to a complete processor that can execute real-world applications.

However, there are some limitations and potential areas for future work. One noteworthy aspect is the current design's support for full 32-bit instructions. The RISC-V architecture includes compressed instructions that are 16 bits in length, aimed at optimizing code density. To achieve compliance with the full RISC-V specification, the CPU would need to be extended to support these compressed instructions, for example the instruction discussed in class "lh", which loads only 16-bits (load halfword). Additionally, our design focuses on a single-cycle CPU, which is relatively straightforward but not as efficient as a pipelined CPU. Future work could explore the transition to a pipelined design, which would further improve instruction throughput and performance. Lastly, more rigorous testing and verification processes could be implemented to ensure the CPU's correct functionality and reliability when executing any complex programs.

CONCLUSION:

[10%] Conclude the lab report, stating what the lab was about. Mention any major implementation issues and summarize your observations and discussion (1 paragraph).

In conclusion, this lab centered on the implementation of the missing Verilog code for the instruction memory module in a single-cycle CPU design. Our primary focus was to rectify this essential component's absence, allowing us to move closer to the realization of a fully functional processor for executing RISC-V instructions. Throughout the lab, we successfully declared, initialized, and integrated the instruction memory, enhancing our understanding of digital circuit design. While our work has been instrumental, there is room for future improvements, such as supporting compressed instructions and exploring pipelined designs. Overall, this lab marks a significant milestone in the development of our CPU, bringing us closer to a comprehensive and efficient computing system.