# CSC 565 - Computer System Architecture: Final Project Report

Due on April 27, 2023 @ 18:00, EST

*Fundamentals of Microprocessor Design*

**Cason Konzer**

April 23, 2023

# Contents

**Introduction** **3**
Requirements . . . 3
Motivation . . . 3

**Fundamentals** **3**
Where I Stood . . . 3
Information Retrieval . . . 4

**Microprocessor Specifications & Design** **4**
Bottom Up Approach . . . 4
Handling Pitfalls . . . 5

**Microprocessor Testing** **5**
The Test Bench . . . 5
Design Changes . . . 5

**Summary** **6**

**Appendix : Tables** **7**

**Appendix : Figures** **8**

# List of Tables

1 Register Specifications. . . . 7
2 IR Encoding. . . . 7
3 Destination Encoding. . . . 7
4 Operand Encoding. . . . 7
5 Operation Encoding. . . . 7

# List of Figures

1 Initial Microprocessor Block Diagram. . . . 8
2 First Revised Microprocessor Block Diagram. . . . 8
3 Second Revised Microprocessor Block Diagram. . . . 9
4 Third Revised Microprocessor Block Diagram. . . . 9
5 Final Microprocessor Block Diagram. . . . 10
6 Initial Timing, with buffer registers. . . . 10
7 Revised Timing, with inserted no-ops. . . . 10

# Introduction

## Requirements

Within the final project for **CSC 565** I was tasked with creating a simple microprocessor on 8 bit registers. This microprocessor was to follow the following specifications:

- ◇ Inputs (data and instructions) are received from outside the processor, computed flags and results are to be stored in a register.
- ◇ The processor takes all operands directly from internal registers.
    - ∗ operands are not to be passed to the ALU from the instruction itself.
- ◇ The processor must support integer subtraction and logical or as operations.
- ◇ The processor must support sign and overflow flags.
    - ∗ flags are internal and must not be output.
- ◇ The destination for all instructions should first be an internal register, and in the (next) clock cycle following the instruction load the computed output should be sent on the output bus.

A series of deliverables were specified to guide students in the project in the following manner:

1. Draft & finalized specifications document containing:
    - ○ Block diagram with input/output pins.
    - ○ Number of registers and their sizes.
    - ○ Instruction set with defined operands.
    - ○ Definition of clock signal.
    - ○ Timing diagram for example instructions.
2. Designed device to the specs in Verilog.
3. Tested and properly functioning device.
4. Final project report.

## Motivation

From the basic specifications and deliverables outlines above I had a solid scope of what was required of me, competencies I had, and competencies I needed to gain. In general this project was self guided and required a proactive student to succeed. For myself, I got an early start and had been following the corse ahead of schedule, thus this was well catered. What I found extremely motivating was the process itself, and getting hands on with the logic of hardware. Working with microprocessor devices, in an environment where they are designed all the way from the hardware and software, to the actual integration, gaining a solid understanding of the lower level workings was critical to me. With the foundation laid, I will now walk my way through each of the deliverables reflecting on progress made, challenges faces, individual learnings, and potential pitfalls.

# Fundamentals

## Where I Stood

Clear from the outset was that I was an incompetent microprocessor designer, and I had much to learn. Many times I have seen colleagues working on wiring diagrams but never myself. My first task at hand was to build a fundamental understanding of the concepts which would be utilized. I started off by identifying my competencies that were built upon prior readings, class lectures, and assignments. Concepts which were understood included:

- ◇ What Registers are.
- ◇ How block diagrams generally looked.
- ◇ Basics of instructions.
- ◇ Flags and what they represent.
- ◇ The stepping form factor of a clock signal.

Unclear to me was:

- ◇ Register functional design.
- ◇ Clock Location.
- ◇ Clock signal generation.
- ◇ Synchronization via clocks.
- ◇ Instruction set encoding.
- ◇ Overflow flag computation.
- ◇ Timing diagrams.
- ◇ Verilog programming.

### Information Retrieval

My initial investigations started with a brief overview of the sign, carry, and overflow flag, followed by more detailed views of their operations [3, 11, 25, 26, 27]. Feeling now more confident in flags I proceeded to dig into registers and how they function [17, 18, 19]. At this point I was still fuzzy on clocks and thus I did some reading up on them as well [4, 23, 22, 28]. Before programming in Verilog I needed a more sound knowledge of combinatorial logic and thus proceeded to get an understanding of what lay inside the blocks which I would be placing in my pinouts [5, 6, 7, 9, 8, 10, 12]. In order to implement my logic in Verilog I would needed to know how to program in it [15, 16]. Last for testing my machine I would need to utilize a scripting language. Most familiar to me is Python, but for bit manipulation in the language I required background [1, 14, 20, 21, 24]. Despite the plethora of cited materia this does not encompass everything and thus is incomplete. The foundation is also well complemented by the professor's (Dr. Zahid Syed's) teachings and the class textbooks [2, 13].

It is with this foundation that I learned the fundamentals of combinatorial logic, flag computation, and register operation. The sign flag I found quite trivial, but an analysis on inputs and outputs provided an elegant way of testing for overflow I was happy to see. I learned about latches and flip flops in registers, which greatly helped me understand the lockstep nature of microprocessors. With this under my belt I was able to start dabbling in Verilog using tools provided by Xilinix and Aldec. I started creating block diagrams and synthesizing them into code to get a feel for the language. I would first draw them out and then port them into the programming environment. I took the sample code from the professor and modified it to similar use cases. I learned the pros and cons of internal and external clocks, and how to read timing diagrams. At this point I was ready to start specifications while designing in the background.

## Microprocessor Specifications & Design

### Bottom Up Approach

Generally speaking, specifications and design were paired tasks. It was specifications which drove initial design, but it was realizations of issues and optimizations in the design which granted revisions to the specifications. The process started with defining the registers, which enabled me to visualize the first linking of the operand registers to the ALU. Form this point it was crucial for me to analyze the inputs and outputs.

How the processor knew data was valid and where to send it was realized by enabling pins and a input data bus. For myself, it was fundamental to work first on a bit level rather than a byte level. My initial diagrams contained 8 separate input wires rather than one input bundle for the data bus. I next made a definitive choice to use an external clock, while in practice I would prefer support for both internal and external clocks, this simplified the design.

I next started to define the instruction set, and its encoding. From here I was able to link the ALU to the instruction register. While at first my block diagram was criticized for being too granular, the bottom up approach I took gave me depth in understanding the microprocessor. This made it extremely easy to generalize my design and block diagram to a more abstract level. Initially I thought that I would have the ALU output the computed results to a designated register, and the flags to a dedicated register as well. Upon further revision I realized that my instruction set needed not all of the bits packed into a byte to encode all possible instructions. Additionally, I realized that given a 2 bit enable input I need not even provide load instructions, but this could rather be encoded into the enable bits. As a result I decided to pack the ALU flags into the already determined instruction register and further simplify the design. In a similar manner, I realized that I need not an ALU output register, but I could instead write back to the operand registers.

### Handling Pitfalls

What occurred as a result was a race condition such that as operand register values were updated, if the instruction register was not flushed, the ALU would continuously perform the instruction on the new register values. In order to mitigate this issue I needed to encode an operation flag into the instruction register which would update with the clock. I decided to leverage an already defined portion of the instruction encoding, the destination encoding. The destitution encoding defined where the ALU was to write back to, being the data bus, and one or both of the operand registers. My solution was to update the instruction register with the destinations encoding '00' which specified only output bus writing, thus mitigating the race condition. This process was done every clock cycle but only if the enable pins had been set to disable instruction loads. In the case that the enable pins were set to load data, regardless of the location, the writing of the ALU to any location was by default disabled. In the end I had created a microprocessor designed to operate in an embedded real-time manner. The hardware requirements were absolutely minimized, as well as the required intermediate buffering registers. The specifications for such a system are realized in the Appendices.

## Microprocessor Testing

### The Test Bench

In testing such a system I started with an iterative approach. I used the professors base code to create simple test vectors, at first by hand. This was sufficient for 2 and 4 bit operands, but became increasingly complex for 8 bit operands, or multiple inputs. Everything at first was designed on the bit level and I soon realized that this made not only my Verilog code overly complex, but also the requirements for testing my models. I was writing then my test vectors though Python, and importing the files into ActiveHDL. I decided at this point it would be beneficial to refactor my code, and in doing so I found a nice repository which extended some of my previous bitwise functions for multiplexing and addition to bytewise [1]. This made my test cases much simpler to write, but still I needed to formulate a mathematical model in a higher level language for comparison. Leveraging the bit manipulation skills I learned in Python I created such models and evolved my test benches to self checking test benches. I randomized operand inputs and extensively tested each instruction to ensure proper functionality. In the end my processor functioned as expected without issue.

### Design Changes

As mentioned above the switch to writing Verilog on bundles of wires (or vectors), rather than individual wires (bits) was a design change driven by my testing. Additionally Testing and analysis of waveforms gave me the realization that within my code there were still some buffer register for the operands when feeding into

[1] `https://github.com/neelkshah/MIPS-Processor`

the ALU. In part this was a countermeasure to at first deal with the race condition, but more generally this was a realization that still I had not a great grasp on programming in Verilog. I continued to explore external resources while taking a Coursera course on the material which gave me a much deeper understanding of the logic Verilog was implementing, tools it had available, and a secondary VHDL perspective. Knowing then the differences between asynchronous and synchronous operations, I again refactored my code to eliminate the temporary registers. In this manner I separated out much of the combinatorial logic into their own modules, which I then linked to the controlling logic. After a final revision I was happy with my design and the timing diagrams it produced, the before and after are shown in Figure 6 and Figure 7. In a similar manner evolution of the design can be seen through the microprocessor's block diagrams Figures [1,2,3,4,5]. The microprocessor was minimal and functional, only the necessary components were used.

## Summary

In summary I learned a plethora about how microprocessors actually work, and how the design process takes place. Not only the technical skills did I manage to gain, but additionally communication, documentation, and planning skills. This project taught me how to keep a well established communication channel with my peers, although my group did eventually dissolve. I managed scheduling meetings with my peers and educating them on the background topics I learned. I took the opportunities to learn from them, and converse on course topics, which helped me to identify my knowledge gaps. I assigned tasks collaboratively and followed up with the members, although most often they had not held up their end of the bargain. I was fast to communicate concerns not only directly to my peers, but also to the professor of the unfolding situation. In doing so I found myself necessarily working ahead and completing the tasks assigned to peers. As two of the students ended up dropping the class, I prevented myself from being in a scramble at the end of the semester. To me, being ahead of the game was necessary, as I never knew when there might be a sudden spike in my day to day employments.

Drawing out the block diagrams, and writing up truth tables proved to be one of my most useful tools. By doing so I was forced to visualize and compute the architecture and modules myself. This process greatly deepened my understanding and allowed me to fluently explain, and show the concepts to group members. Documentation for my background resources was shared and recommended, alternative resources were sought for struggles applying specifically to the peers and not myself. In citing my references I realized the importance of even more extensive documentation, as there were many valuable resources I found to be 'common knowledge' which in any case are due credit. I learned how to program in Verilog, and design a system from the bottom up, test it and verify it's functionality. This has been by far my favorite class, and the project was a pleasure. I am grateful that I can now have more technical conversations on computing architecture, and speak from this experience.

# Appendix : Tables

| Register | Description |
|---|---|
| IR | Instruction Register + Flags |
| R0 | Operand Register 0 |
| R1 | Operand Register 1 |

Table 1: Register Specifications.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Encoding | Overflow Flag | Sign Flag | OR OpCode | SUB OpCode | Operand 1 | Operand 0 | Destination 1 | Destination 0 |

Table 2: IR Encoding.

| IR Bit 1 | IR Bit 0 | Write Back Location |
|---|---|---|
| 0 | 0 | Bus |
| 0 | 1 | R0 & Bus |
| 1 | 0 | R1 & Bus |
| 1 | 1 | R0 & R1 & Bus |

Table 3: Destination Encoding.

| IR Bit 5 | IR Bit 4 | Operands |
|---|---|---|
| 0 | 0 | R0, R0 |
| 0 | 1 | R1, R0 |
| 1 | 0 | R0, R1 |
| 1 | 1 | R1, R1 |

Table 4: Operand Encoding.

| IR Bit 5 | IR Bit 4 | ALU Operation |
|---|---|---|
| 0 | 0 | No-Op |
| 0 | 1 | SUB |
| 1 | 0 | OR |
| 1 | 1 | No-Op |

Table 5: Operation Encoding.

# Appendix : Figures

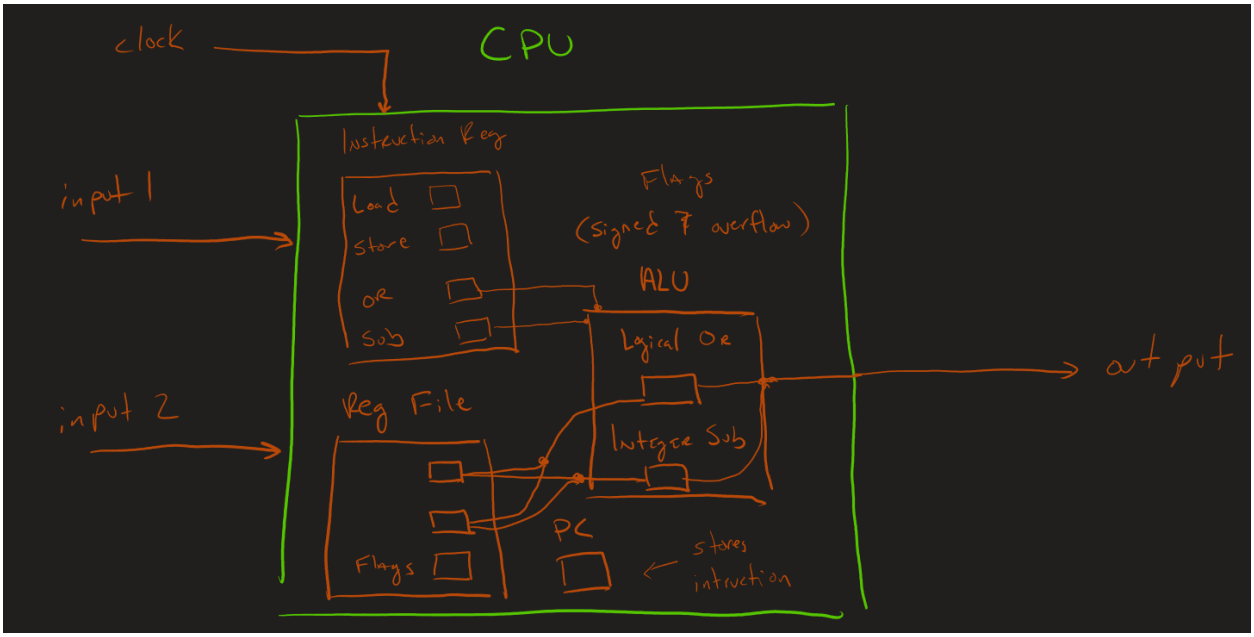


Figure 1: Initial Microprocessor Block Diagram.

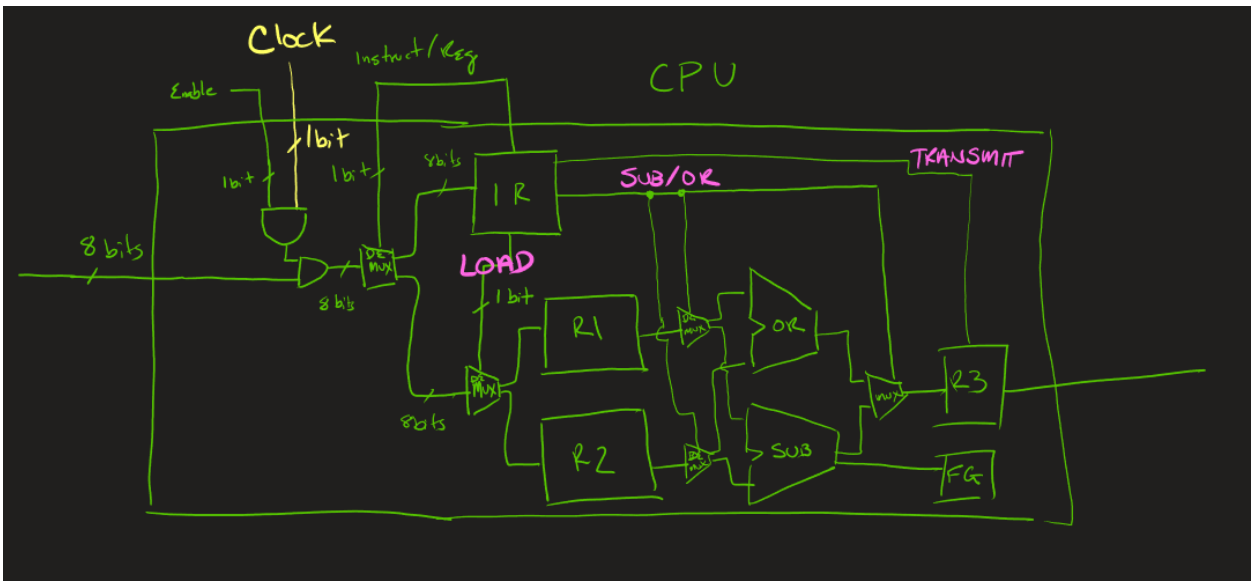


Figure 2: First Revised Microprocessor Block Diagram.

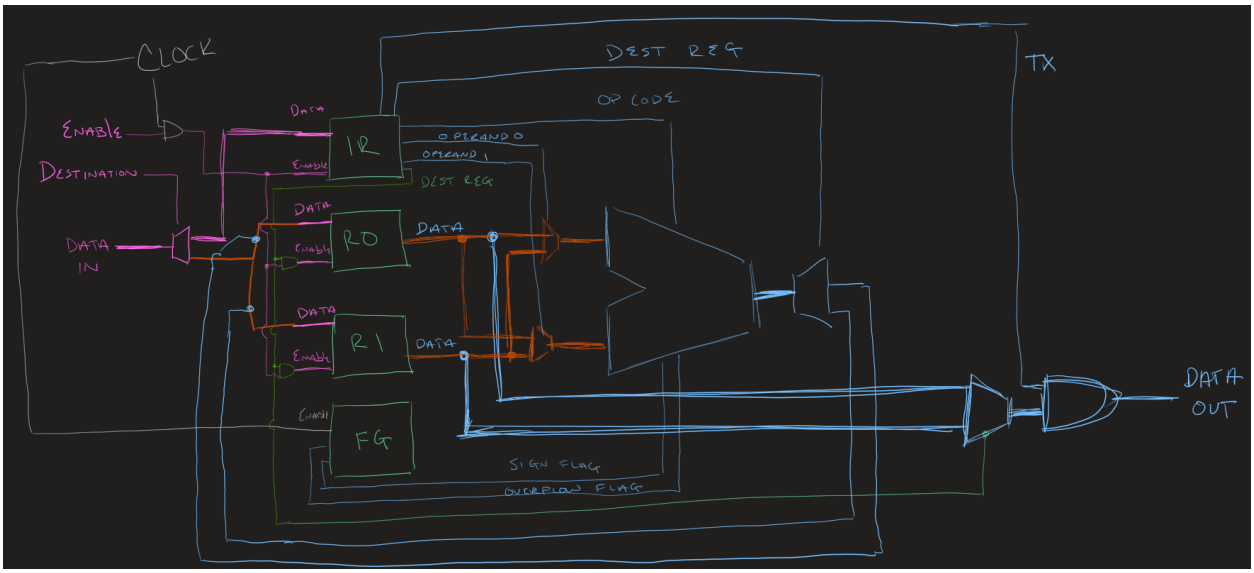


Figure 3: Second Revised Microprocessor Block Diagram.

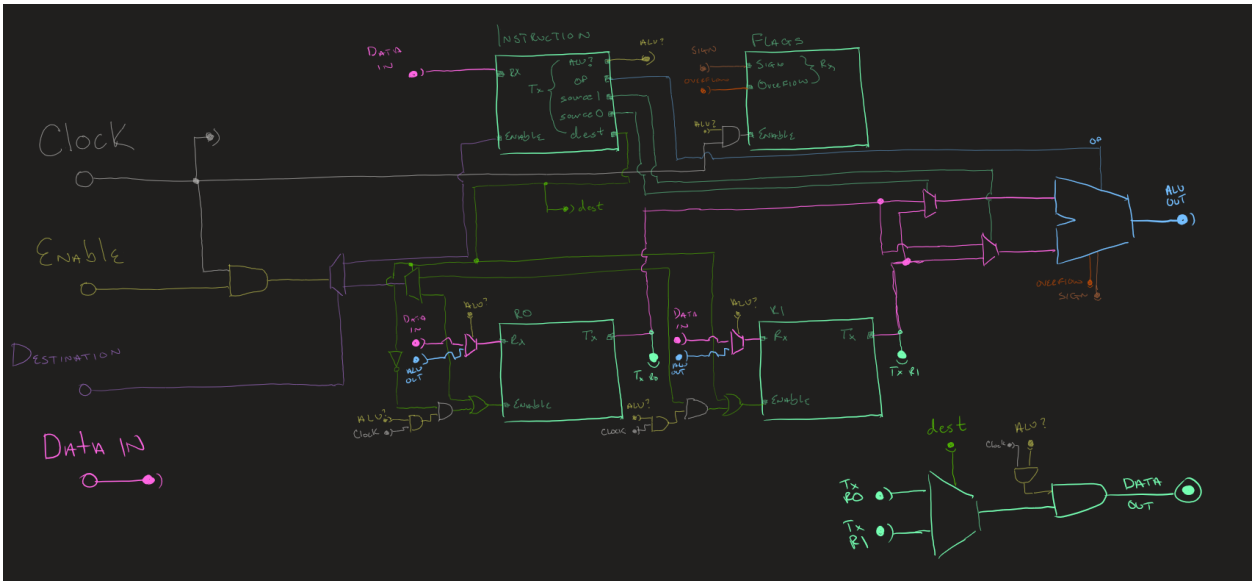


Figure 4: Third Revised Microprocessor Block Diagram.

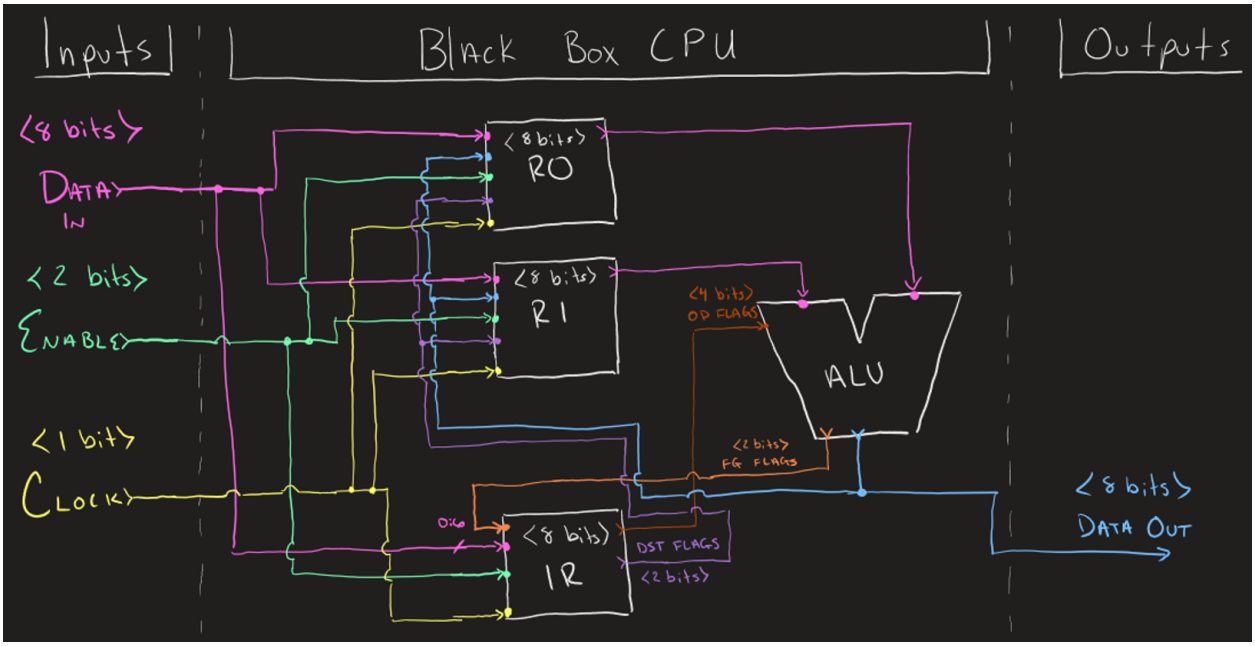


Figure 5: Final Microprocessor Block Diagram.


Figure 6: Initial Timing, with buffer registers.

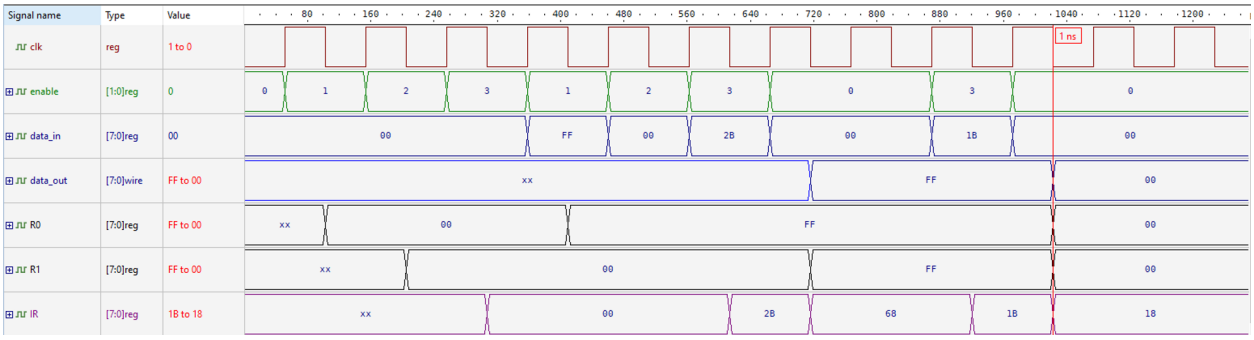


Figure 7: Revised Timing, with inserted no-ops.

# References

[1] (akku), S. C. Efficient method for 2's complement of a binary string. https://www.geeksforgeeks.org/efficient-method-2s-complement-binary-string/, July 2020.

[2] Bryant, R. E., and O'Hallaron, D. R. *Computer Systems: A Programmer's Perspective*, 3rd, global ed. Pearson, 2019.

[3] DarkLink, Millikan, R., and Lazcol. Binary arithmetic - overflow and carryout at same time? https://stackoverflow.com/questions/19301498/carry-flag-auxiliary-flag-and-overflow-flag-in-assembly, September 2020.

[4] Dipankar, Charife, M., Steinbauer, M., and Ostermiller, S. What is the difference between internal and external clock synchronization in distributed systems? https://stackoverflow.com/questions/36446807/what-is-the-difference-between-internal-and-external-clock-synchronization-in-di, January 2023.

[5] Electronics Tutorials. Binary adder. https://www.electronics-tutorials.ws/combination/comb_7.html.

[6] Electronics Tutorials. Binary subtractor. https://www.electronics-tutorials.ws/combination/binary-subtractor.html.

[7] Electronics Tutorials. Combinational logic circuits. https://www.electronics-tutorials.ws/combination/comb_1.html.

[8] Electronics Tutorials. The demultiplexer. https://www.electronics-tutorials.ws/combination/comb_3.html.

[9] Electronics Tutorials. Digital comparator. https://www.electronics-tutorials.ws/combination/comb_8.html.

[10] Electronics Tutorials. The multiplexer. https://www.electronics-tutorials.ws/combination/comb_2.html.

[11] Embedded Systems and Deep Learning. Lecture 3: Overflow flag for signed addition and subtraction. https://www.youtube.com/watch?v=BIn6iyYIGio, October 2016.

[12] Harries, I. Arithmetic operations on binary numbers. https://www.doc.ic.ac.uk/ eedwards/compsys/arithmetic/index.html, October 2021.

[13] Hennessy, J. L., and Patterson, D. A. *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2017.

[14] Jim, erikbstack, travc, Fritz, Écorchard, G., Griffiths, S., and Rooy, J. L. Two's complement in python. https://stackoverflow.com/questions/1604464/twos-complement-in-python, Febuary 2023.

[15] Meher Krishna Patel. Testbenches. https://www.coursera.org/learn/fpga-hardware-description-languages#syllabus.

[16] Meher Krishna Patel. Testbenches. https://verilogguide.readthedocs.io/en/latest/verilog/testbench.html, 2017.

[17] Neso Academy. Difference between latch and flip flop. https://www.youtube.com/watch?v=m1QBxTeVaNs, Febuary 2015.

[18] Neso Academy. Sr latch | nor and nand sr latch. https://www.youtube.com/watch?v=kt8d3CYWGH4, April 2015.

[19] Neso Academy. Triggering methods in flip flops. https://www.youtube.com/watch?v=Pi_MHyMoenA, Febuary 2015.

[20] pranjalmittal. Bitwiseoperators. https://wiki.python.org/moin/BitwiseOperators, July 2017.

[21] sifferman. Two's complement using only logic gates. https://cs.stackexchange.com/questions/51034/twos-complement-using-only-logic-gates, August 2017.

[22] SimonBarker, Lathrop, O., stevenvh, Passerby, and Johan. Internal or external oscillator. https://electronics.stackexchange.com/questions/15455/internal-or-external-oscillator, June 2020.

[23] Timothy Scherr and Benjamin Spriggs. Hardware description languages for fpga design. https://www.youtube.com/watch?v=wx0NyUfpm48, Febuary 2015.

[24] Trivedi, U. 1's and 2's complement of a binary number. https://www.geeksforgeeks.org/1s-2s-complement-binary-number/?ref=lbp, December 2022.

[25] user2322960, Cordes, P., dark_knight, and Walenciak, M. Carry flag, auxiliary flag and overflow flag in assembly. https://stackoverflow.com/questions/19301498/carry-flag-auxiliary-flag-and-overflow-flag-in-assembly, September 2020.

[26] Wikipedia. Negative flag. https://en.wikipedia.org/wiki/Negative_flag, December 2022.

[27] Wikipedia. Overflow flag. https://en.wikipedia.org/wiki/Overflow_flag, October 2022.

[28] Wikipedia. Clock signal. https://en.wikipedia.org/wiki/Clock_signal, January 2023.