# **CprE 381: Computer Organization and Assembly-Level Programming**

# **Project Part 1 Report**

Team Members: **Jayson Acosta, Parnika Dasgupta, Nakota Clark**

## Project Teams Group #: **Section D\_01**

***Refer to the highlighted language in the project 1 instruction for the context of the following questions****.*

[Part 1 (d)] Include your final MIPS processor schematic in your lab report.

[Part 2 (a.i)] Create a spreadsheet detailing the list of *M* instructions to be supported in your project alongside their binary opcodes and funct fields, if applicable. Create a separate column for each binary bit. Inside this spreadsheet, create a new column for the *N* control signals needed by your datapath implementation. The end result should be an *N*\**M* table where each row corresponds to the output of the control logic module for a given instruction.

**Finished in** [**Proj1\_control\_signals.xlsx**](https://docs.google.com/spreadsheets/d/1onvvnxHIHOjwQesYKJHIwepPtgYyOp8t/edit?usp=drive_link&ouid=102182269893109079723&rtpof=true&sd=true)

[Part 2 (a.ii)] Implement the control logic module using whatever method and coding style you prefer. Create a testbench to test this module individually, and show that your output matches the expected control signals from problem 1(a).

[Part 2 (b.i)] What are the control flow possibilities that your instruction fetch logic must support? Describe these possibilities as a function of the different control flow-related instructions you are required to implement.

**Our fetch logic supports the following control flow possibilities:**

* **Sequential execution: PC = PC + 4**
* **Branch instructions (beq, bne): PC = PC + 4 + (sign-extended immediate << 2)**
* **Jump instructions (j, jal): PC = (PC + 4)[31:28] concatenated with (address << 2)**
* **Jump Register (jr): PC = register value**
* **These correspond to the MIPS branch (beq, bne), jump (j, jal), and jump register (jr) instructions.**

[Part 2 (b.ii)] Draw a schematic for the instruction fetch logic and any other datapath modifications needed for control flow instructions. What additional control signals are needed?

[Part 2 (b.iii)] Implement your new instruction fetch logic using VHDL. Use Modelsim to test your design thoroughly to make sure it is working as expected. Describe how the execution of the control flow possibilities corresponds to the Modelsim waveforms in your writeup.

[Part 2 (c.i.1)] Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction?

The difference between srl and sra is how it handles the sign bit. Logical shifts don’t care about the numerical value, and so it shifts all the bits and place 0’s in the new bits. Arithmetic shifts treat the values as numbers, and look at the sign bit to decide whether to shift in 0’s or 1’s. Sla does not exist because shifting to the left shifts in values to the least significant bits. Looking at the sign bit to decide which value to shift into these bits will not preserve the numerical value, and instead will make the number entirely different than expected. This means you should always shift in 0’s, so sla would be the same as sll, and would be redundant to add separately.

Example of sla being incorrect:

-1 -4

sll 0b1111, 2 => 0b1100

-1 -1

sla 0b1111,2 => 0b1111

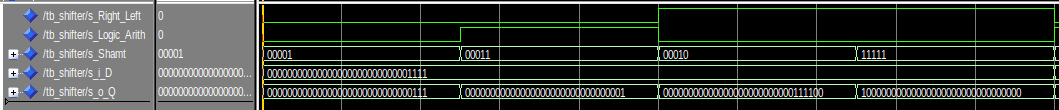
[Part 2 (c.i.2)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.

My code implements both arithmetic and logical operations by using a control signal and if/elsif statements. The control bit is Logic\_Arith; 0 means logical and 1 means arithmetic. This value decides which branch of the if/elsif to take. For logical shifts, I shift the data and treat it as an unsigned number, so that it shifts in 0’s. Arithmetic treats the data as a signed number, so it will shift in 0’s or 1’s depending on the sign bit.

[Part 2 (c.i.3)] In your writeup, explain how the right barrel shifter above can be enhanced to also support left shifting operations.

In order to support left shift operations, I will just add a new control signal Right\_Left, and this will decide whether to do a left or right shift. I will concatenate this bit with the Logic\_Arith bit and use that combined signal as the control for the if/elsif. If the Right\_Left bit is 1, it doesn’t matter what the Logic\_Arith bit is. I’ll only need to add 1 more branch to shift the data to the left.

[Part 2 (c.i.4)] Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.



This image shows the shifter working properly for basic cases. I kept the input the same for all of them to make it easier for me to compare the results. The data in was always 0b00000000000000000000000000001111. The first operation was a srl operation with a shamt of 1. The result matches the expected result of 0b00000000000000000000000000000111. The second was a sra operation with a shamt of 3. It gave the expected result of 0b00000000000000000000000000000001. Then came a sll with a shamt of 2. It gave the expected result of 0b00000000000000000000000000111100. Finally there is a sll with a shamt of 31. It gave the expected result of 0b10000000000000000000000000000000. This shows that the operations are working as expected.

[Part 2 (c.ii.1)] In your writeup, briefly describe your design approach, including any resources you used to choose or implement the design. Include at least one design decision you had to make.

[Part 2 (c.ii.2)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

[Part 2 (c.iii)] Draw a simplified, high-level schematic for the 32-bit ALU. Consider the following questions: how is Overflow calculated? How is Zero calculated? How is slt implemented?

[Part 2 (c.v)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

[Part 2 (c.viii)] justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs functioning.

[Part 3] In your writeup, show the Modelsim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

[Part 3 (a)] Create a test application that makes use of every required arithmetic/logical instruction at least once. The application need not perform any particularly useful task, but it should demonstrate the full functionality of the processor (e.g., sequences of many instructions executed sequentially, 1 per cycle while data written into registers can be effectively retrieved and used by later instructions). Name this file Proj1\_base\_test.s.

[Part 3 (b)] Create and test an application which uses each of the required control-flow instructions and has a call depth of at least 5 (i.e., the number of activation records on the stack is at least 4). Name this file Proj1\_cf\_test.s.

[Part 3 (c)] Create and test an application that sorts an array with *N* elements using the BubbleSort algorithm ([link](http://en.wikipedia.org/wiki/Bubble_sort)). Name this file Proj1\_bubblesort.s.

[Part 4] report the maximum frequency your processor can run at and determine what your critical path is. Draw this critical path on top of your top-level schematics. What components would you focus on to improve the frequency?