# CprE 381: Computer Organization and Assembly-Level Programming

# Project Part 2 Report

Team Members: Evan Shiber

Josh Arceo

Joseph Barnes III

## Project Team Group #: Sec C 03

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

[1.a] Come up with a global list of the datapath values and control signals that are required during each pipeline stage.

[1.b.ii] high-level schematic drawing of the interconnection between components.

A diagram of software schematic

Description automatically generated

[1.c.i] include an annotated waveform in your writeup and provide a short discussion of result correctness.

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

[1.c.ii] Include an annotated waveform in your writeup of two iterations or recursions of these programs executing correctly and provide a short discussion of result correctness. In your waveform and annotation, provide 3 different examples (at least one data-flow and one control-flow) of where you did not have to use the maximum number of NOPs.

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

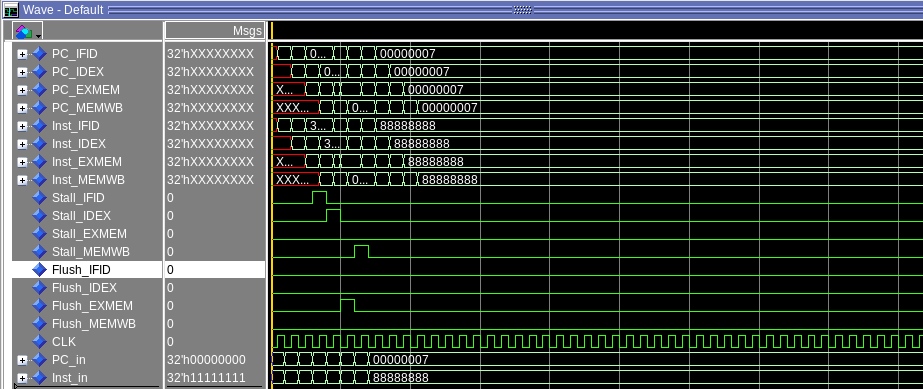
A screen shot of a computer

Description automatically generated

[1.d] report the maximum frequency your software-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).

[2.a.ii] Draw a simple schematic showing how you could implement stalling and flushing operations given an ideal N-bit register.

[2.a.iii] Create a testbench that instantiates all four of the registers in a single design. Show that values that are stored in the initial IF/ID register are available as expected four cycles later, and that new values can be inserted into the pipeline every single cycle. Most importantly, this testbench should also test that each pipeline register can be individually stalled or flushed.



[2.b.i] list which instructions produce values, and what signals (i.e., bus names) in the pipeline these correspond to.

[2.b.ii] List which of these same instructions consume values, and what signals in the pipeline these correspond to.

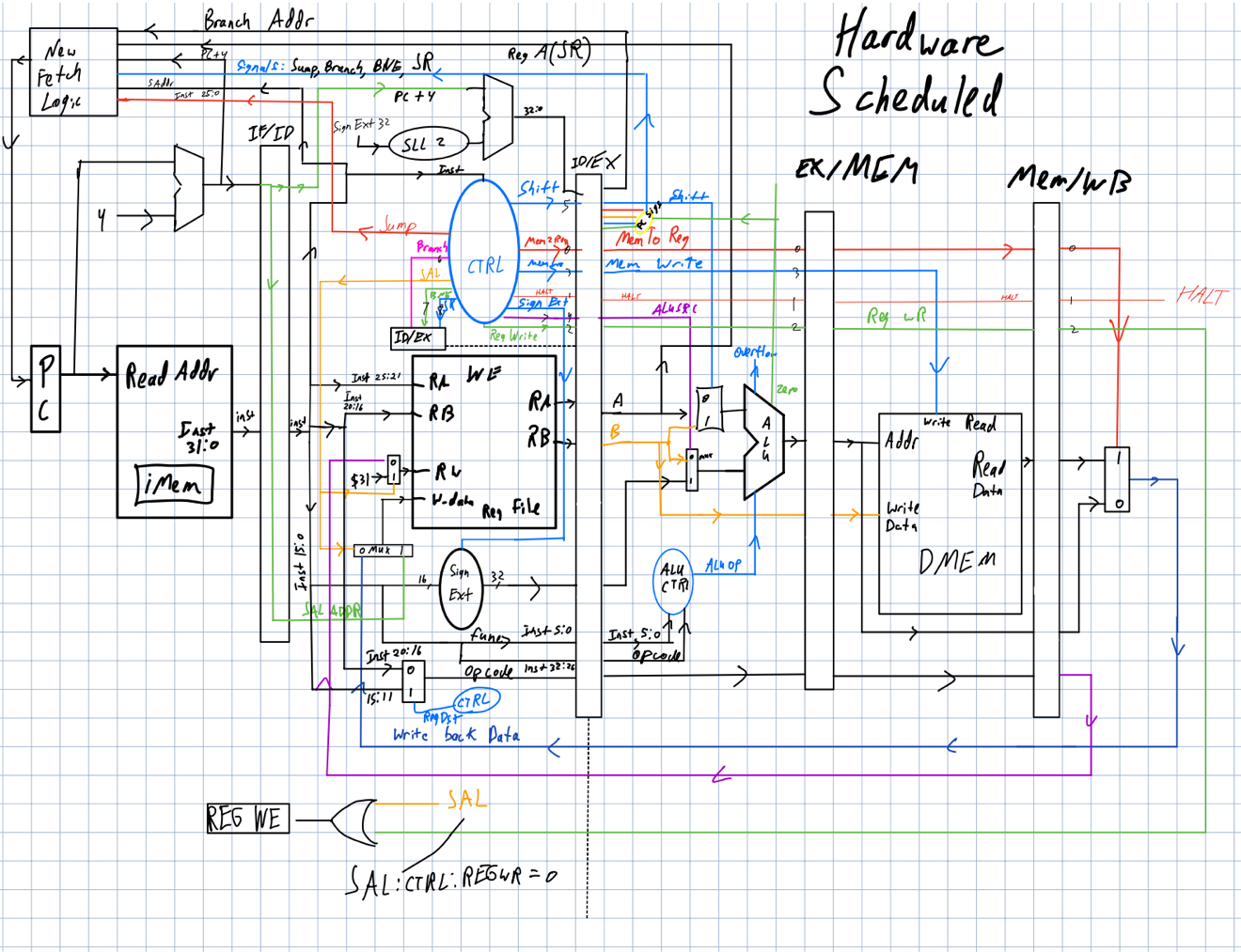
[2.b.iii] generalized list of potential data dependencies. From this generalized list, select those dependencies that can be forwarded (write down the corresponding pipeline stages that will be forwarding and receiving the data), and those dependencies that will require hazard stalls.

[2.b.iv] global list of the datapath values and control signals that are required during each pipeline stage

[2.c.i] list all instructions that may result in a non-sequential PC update and in which pipeline stage that update occurs.

[2.c.ii] For these instructions, list which stages need to be stalled and which stages need to be squashed/flushed relative to the stage each of these instructions is in.

[2.d] implement the hardware-scheduled pipeline using only structural VHDL. As with the previous processors that you have implemented, start with a high-level schematic drawing of the interconnection between components.



[2.e – i, ii, and iii] 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.

[2.e.i] Create a spreadsheet to track these cases and justify the coverage of your testing approach. Include this spreadsheet in your report as a table.

[2.e.ii] Create a spreadsheet to track these cases and justify the coverage of your testing approach. Include this spreadsheet in your report as a table.

[2.f] report the maximum frequency your hardware-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).