CprE 3810: Computer Organization and Assembly-Level Programming

Project Part 2 Report

Team Members: James Gaule

Andy Eslick

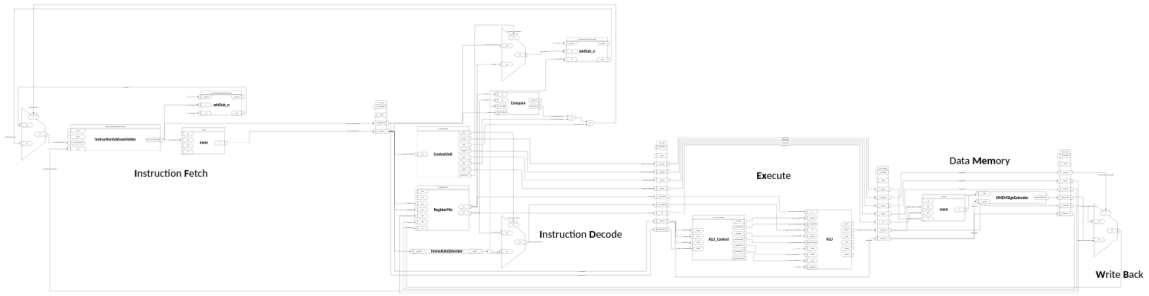
Project Teams Group #: D_05

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.

IF → ID	ID → EX	EX → MEM	MEM → WB
ProgramCounter	Mem_WE	Mem_WE	MemToReg
Instruction	MemToReg	MemToReg	Reg_WE
	Reg_WE	Reg_WE	HaltProg
	HaltProg	HaltProg	DMem_Output
	ALU_Operand1	RS2	ALU_Output
	ALU_Operand2	ALU_Output	Instruction
	RS2	Instruction	
	Instruction		
	ProgramCounter		

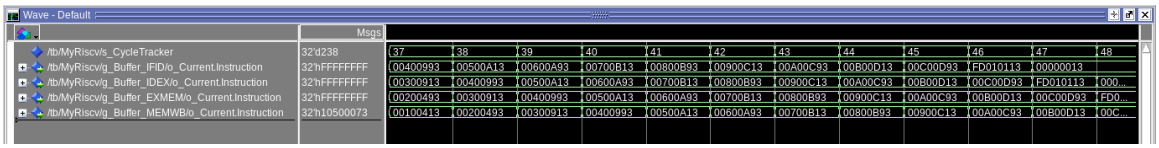
[1.b.ii] high-level schematic drawing of the interconnection between components.



View the full diagram here:
https://iowastate-my.sharepoint.com/:u:/g/personal/esliandy_iastate_edu/IQAAiTNgo3T9RaB7BBPnPt7BAeMyRFAeHUjLhbH9OaDT_Yc?e=tu7gob

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

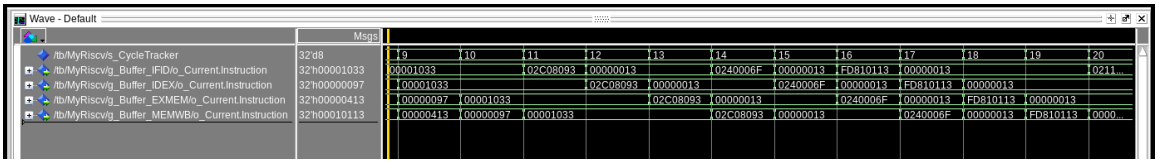
Testbench Waveform



In this segment of the testbench execution, the program is running through a series of addi instructions, each directed to a different register. Because of this, few data hazards currently exist, and thus only one nop needed to be inserted (in cycle 47)

[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.

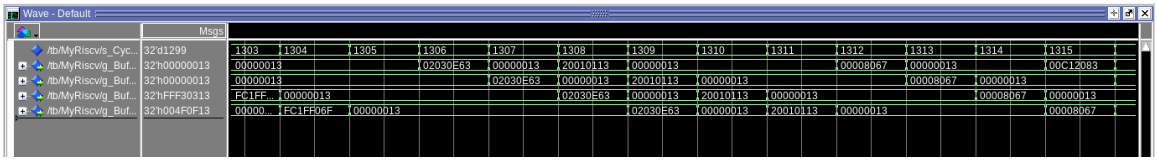
Mergesort Waveform 1



In this portion of the waveform, the function performs a jal function, transitioning from initial setup to the recursive partitioning portions of the mergesort function.

A JAL function is fetched in cycle 14, with a nop inserted immediately after. As the program counter is altered during the ID stage, only one nop must be inserted afterwards to prevent a control-flow hazard.

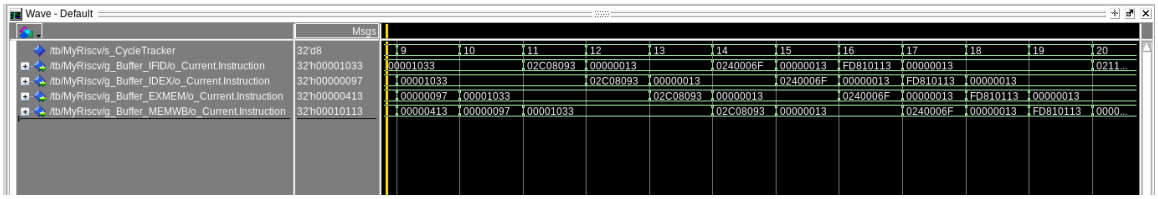
Mergesort Waveform 2



In this portion of the waveform, the program correctly determines that the array merge is completed, and thus triggers a branch back to the main program body.

As branch and jump functions trigger in the ID stage in the pipeline, the function path is decided by the time the branch function reaches the execution stage. Thus, when executing the branch instruction starting at cycle 1306, only one insertion must be added to prevent a control-flow hazard.

Grendel Waveform



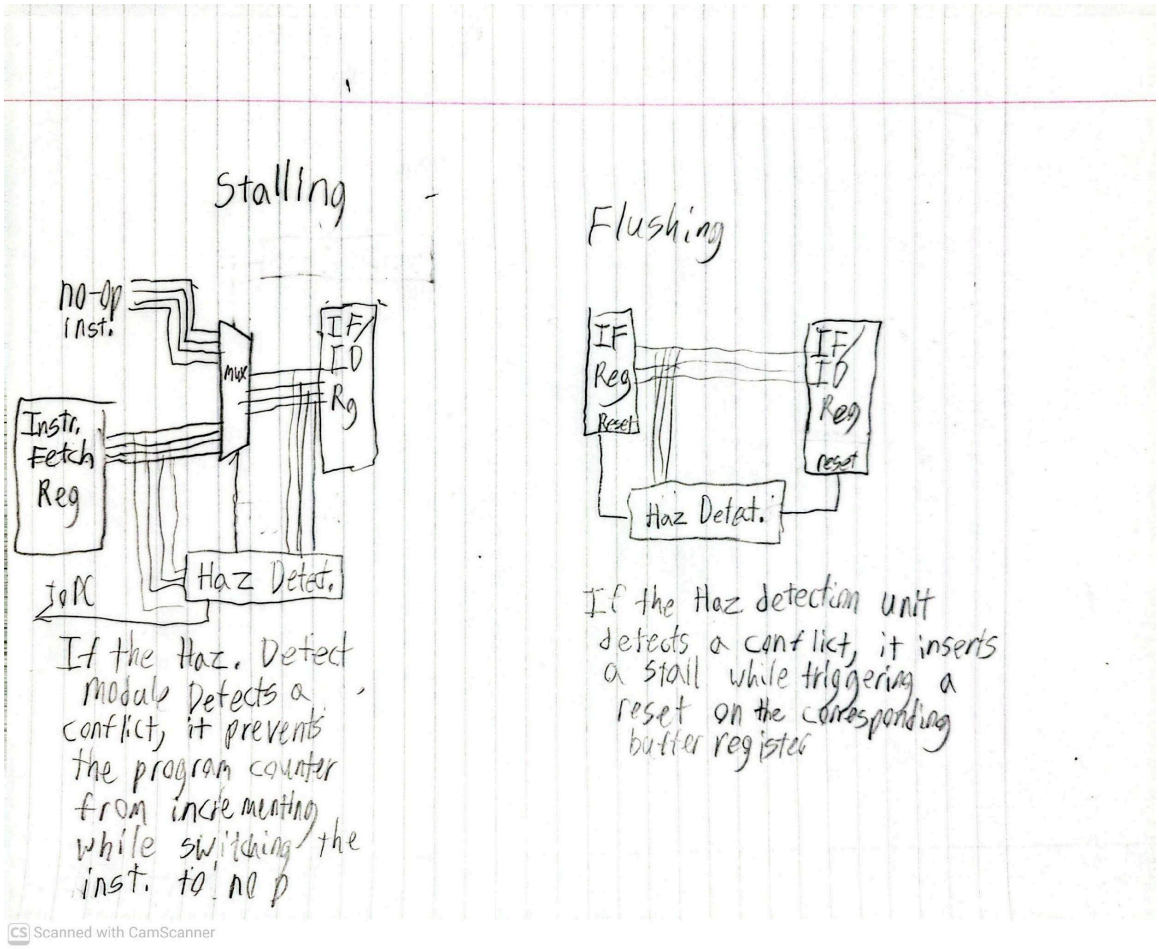
Starting at clock cycle 12, two nops are inserted into the pipeline prior to a JAL command. While these are necessary to ensure that the pipeline is clear following the jump (which may call these values), the jump itself takes a cycle to complete. Thus, the data-flow hazard can be prevented with only two nops.

[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).

Maximum Frequency : 56.42 Megahertz
Critical Path :
IDEX Buffer
ALU Control
ALU AddSub Ripple Adder
ALU Internal Module Select Mux
EXMEM Buffer

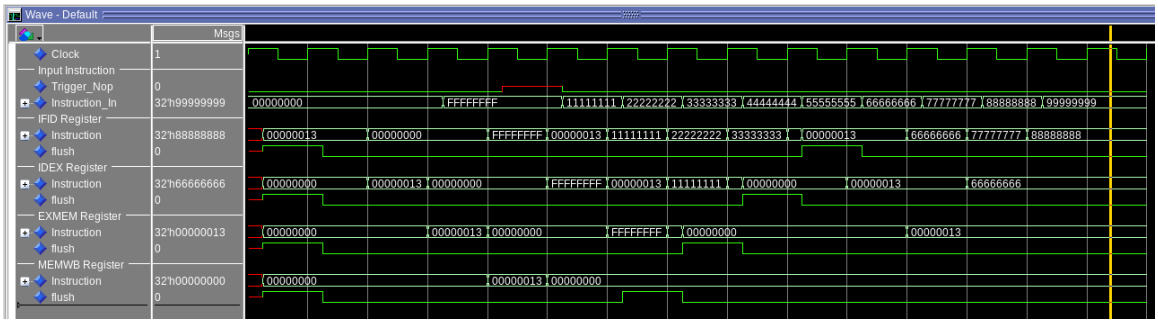
Our Execute stage is the limiting factor for our software scheduled pipeline.

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



TODO

[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.



This waveform first demonstrates the flow of instructions through the pipeline, then flushes every register, then proceeds to issue stall commands to each of the registers.

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

R type, I type, S type, and U type produce ALU Operand values, which map to ALU Operand 1 and ALU Operand 2.

R type, I type, S type, and U type produce an ALU result, which maps to ALU_Output in the pipeline. As the addition for jump and branch functions is performed in a separate component, they do not create this signal.

I load and S instructions produce a 5 bit recording the memory address location, which maps to the RS2 bus in the pipeline.

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

ALU_Output is consumed in the execution stage

Instructions that produce ALU operands consume these in the execution stage.

I load and S instructions consume the RS2 signal in the mem stage.

[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.

Generally dependencies can be forwarded when the prior instruction produces an RD mapped to the same register consumed by a RS1 or RS2 value. One major exceptions are load type instructions (which do not produce a viable RD value until the mem stage)

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

IF → ID	ID → EX	EX → MEM	MEM → WB
ProgramCounter	Mem_WE	Mem_WE	MemToReg
Instruction	MemToReg	MemToReg	Reg_WE
i_Reset	Reg_WE	Reg_WE	HaltProg
i_NOP	HaltProg	HaltProg	DMem_Output
	ALU_Operand1	RS2	ALU_Output
	ALU_Operand2	ALU_Output	Instruction
	RS2	Instruction	
	Instruction		
	ProgramCounter		

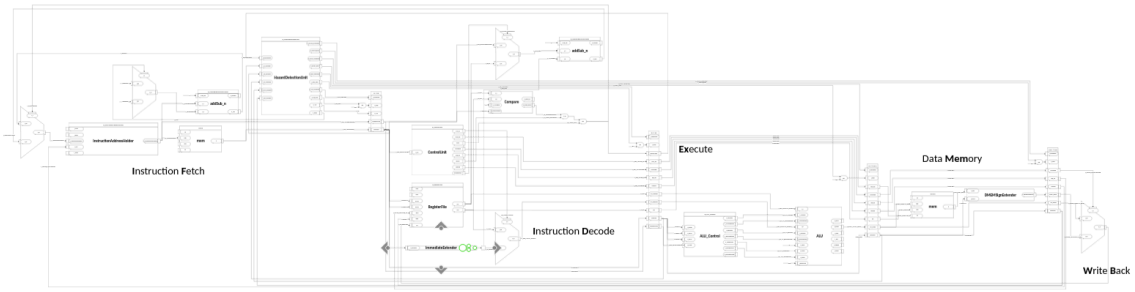
[2.c.i] list all instructions that may result in a non-sequential PC update and in which pipeline stage that update occurs.

Instruction	Program Counter Update Stage
beq	EX
bne	EX
blt	EX
bge	EX
bltu	EX
bgeu	EX
jal	EX
jalr	EX

[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.

Instruction	Stalled	Flushed
beq	IF	ID
bne	IF	ID
blt	IF	ID
bge	IF	ID
bltu	IF	ID
bgeu	IF	ID
jal	IF	ID
jalr	IF	ID

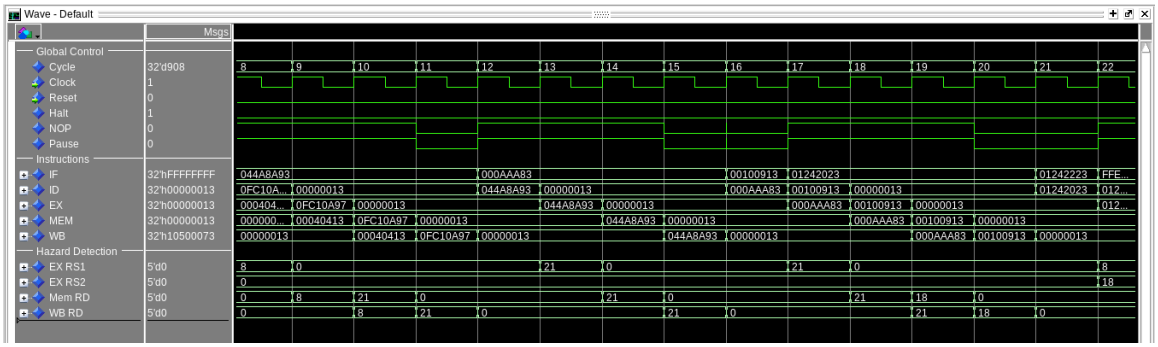
[2.d] implement the hardware-scheduled pipeline using only structural VHDL. As with the previous processors that you have implemented, start with a high-level schematic drawing of the interconnection between components.



View the full diagram here:
https://iowastate-my.sharepoint.com/:u:/g/personal/esliandy_iastate_edu/IQDs8JdA_jrSTooyfvUJi39XAf79lllUsnsDqnXST-6N5Yo

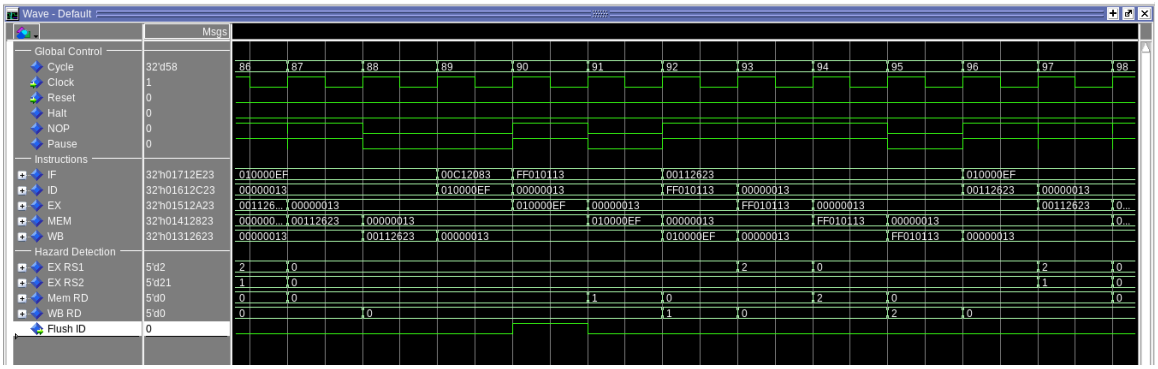
[2.e – i, ii, and iii] In your writeup, show the QuestaSim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

i. Dataflow Hazard



This waveform demonstrates that the hazard detection notes conflicts, directing the IF buffer to stall and insert a nop instruction until this hazard has passed. As data forwarding is not yet working, the processor can only use stalls to prevent data conflicts.

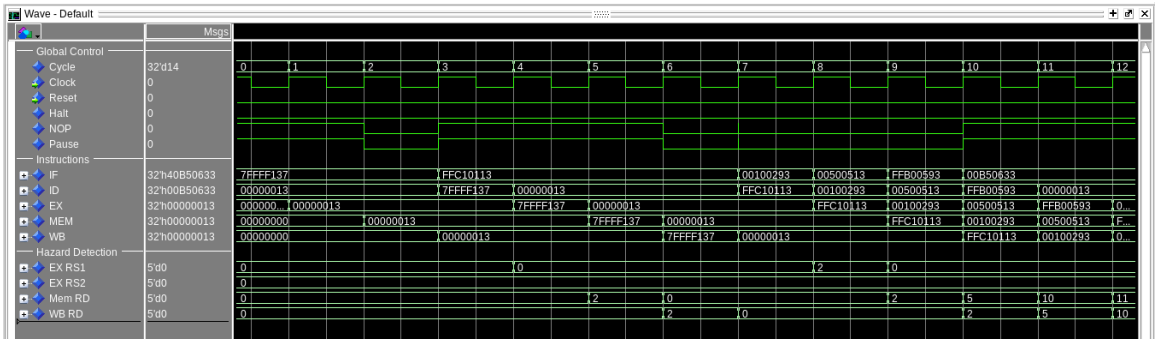
ii. Control Flow Hazards



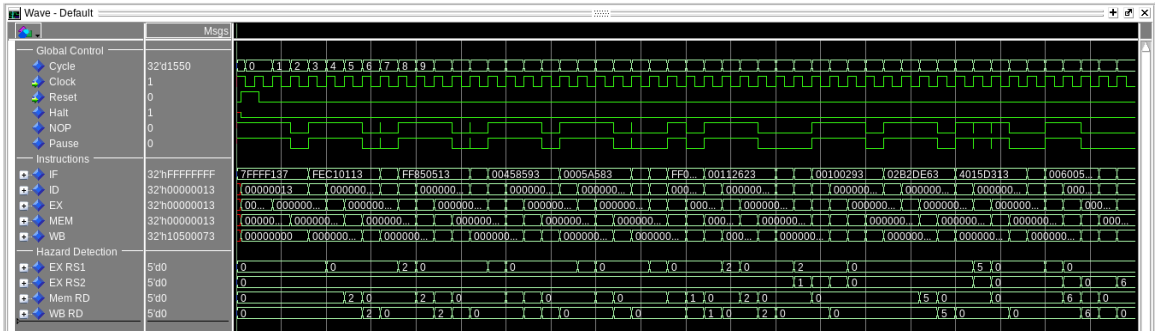
This waveform demonstrates how the hazard detection handles control flow hazards. Starting with the JAL function at cycle 67, the hazard detection unit stalls the IF register, then flushes register ID.

iii. Single Cycle test applications

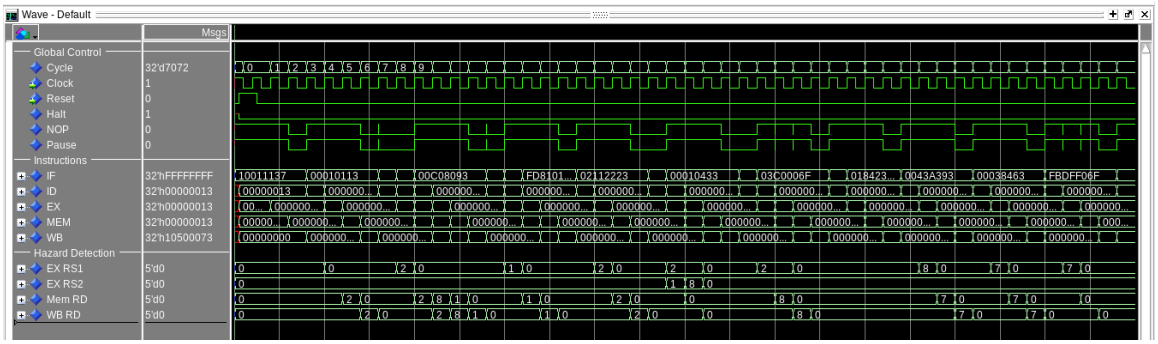
Testbench



Mergesort



Grendel



[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.

Instruction Type	Produces RD value
Arithmetic	Yes
Immediate Arithmetic	Yes
Load	Only in Writeback stage
Store	No
Branch	No
Jump and Link	Yes
Jump and Link Return	Yes
Lui (Load Upper Immediate)	Yes
Auipc (Add upper immediate to pc)	No
Environment	No

[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.

Instruction Type	Can consume in RS1	Can consume in RS2
Arithmetic	Yes	Yes
Immediate Arithmetic	Yes	No
Load	Yes	No
Store	Yes	Yes
Branch	No	No
Jump and Link	Yes	No
Jump and Link Return	Yes	No
Lui (Load Upper Immediate)	No	No
Auipc (Add upper immediate to pc)	No	No
Environment	No	No

[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).

As the hardware-scheduled processor continues to have errors in the forwarding module, the current frequency is based on purely nop insertion. Thus, the maximum frequency is 51.49mhz