CprE 381: Computer Organization and Assembly-Level Programming

Project Part 2 Report

Team Members: Riley Lawson

Long Zeng

Project Teams Group #: 2

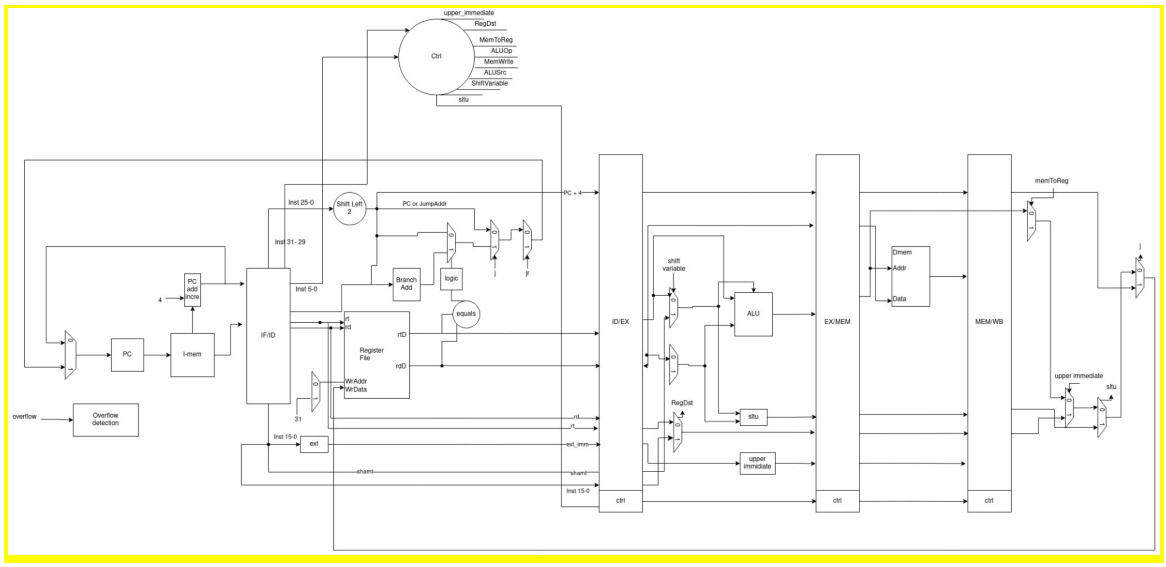
Refer to the highlighted language in the project 1 instruction for the context of the following questions.

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

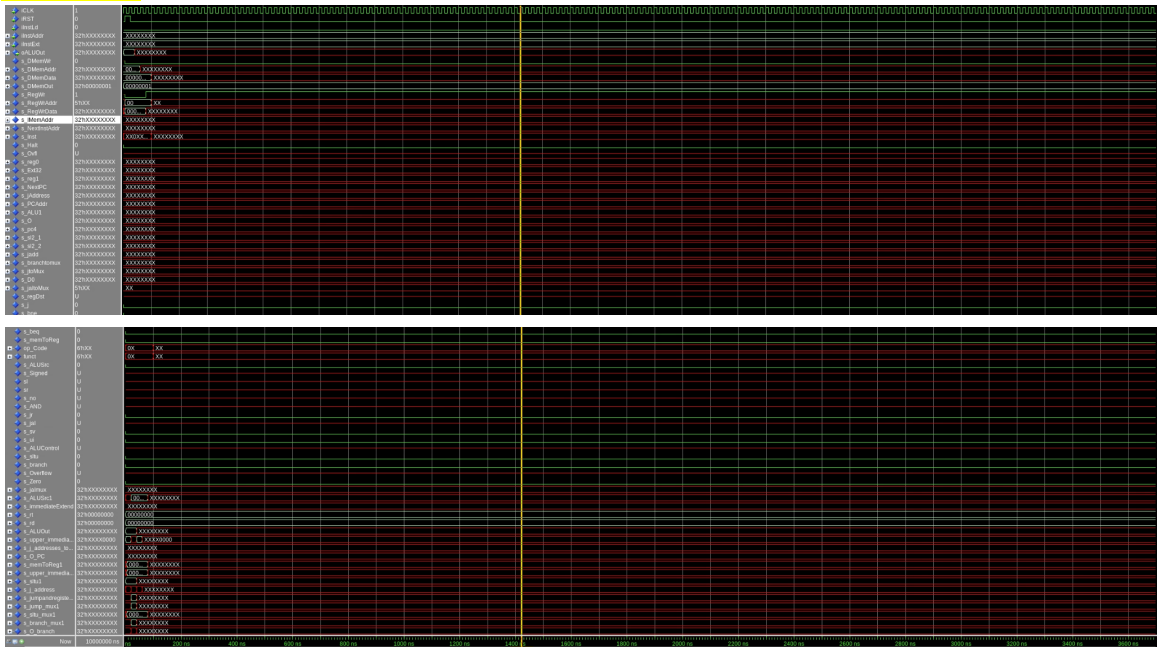
Datapath values:
IF: program counter
ID: program counter, instruction(op code, function code), rt_read, rd_read, data_write, immediate
EX: program counter, control_signal, rt_read, rd_read, signed_immediate, immediate
MEM: program counter, control_signal, r_zero, alu_out, data_read, immediate
WB: control_signal, data_read, alu_out, immediate

Control Signals:
IF→ID:s_Inst,s_ALUSrc,s_ALUControl,s_regDst,s_DMemWr,s_branch,s_memtoReg,s_registerWrite_IFID
ID→EX:s_Inst,s_ALUSrc,s_ALUControl,s_regDst,s_DMemWr,s_branch,s_memtoReg,s_registerWrite_IFID
EX→MEM:s_DMemWr,s_branch,s_memtoReg,s_registerWrite_IFID
MEM→WB:s_memtoReg,s_registerWrite_IFID

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



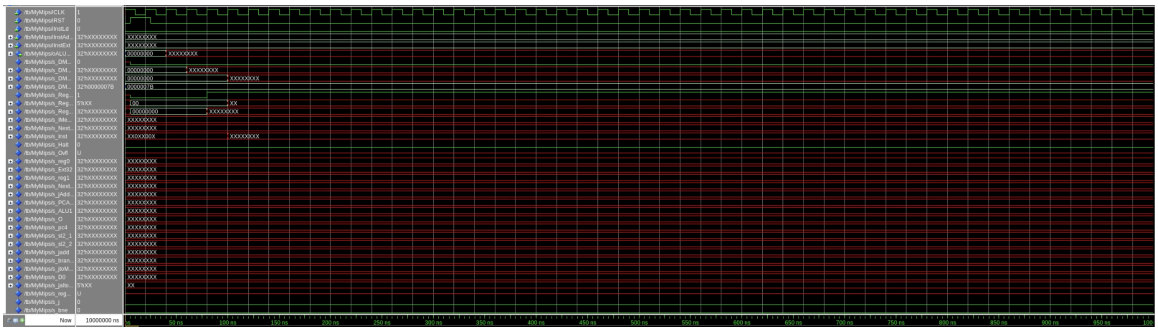
[1.c.i] include an annotated waveform in your writeup and provide a short discussion of result correctness.

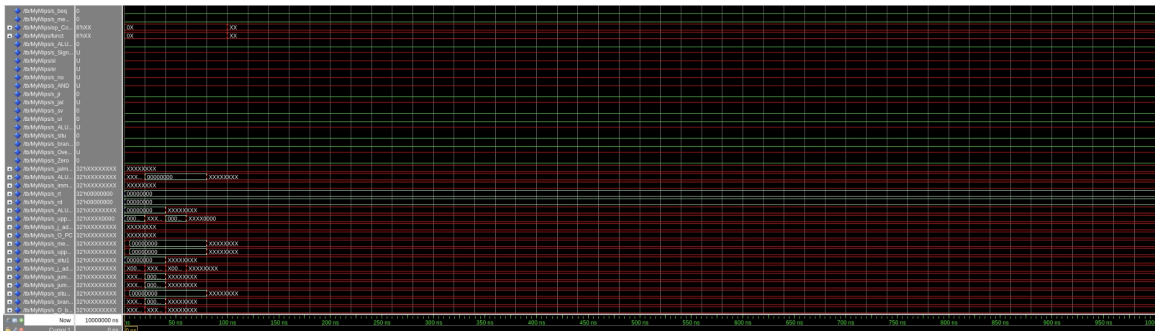


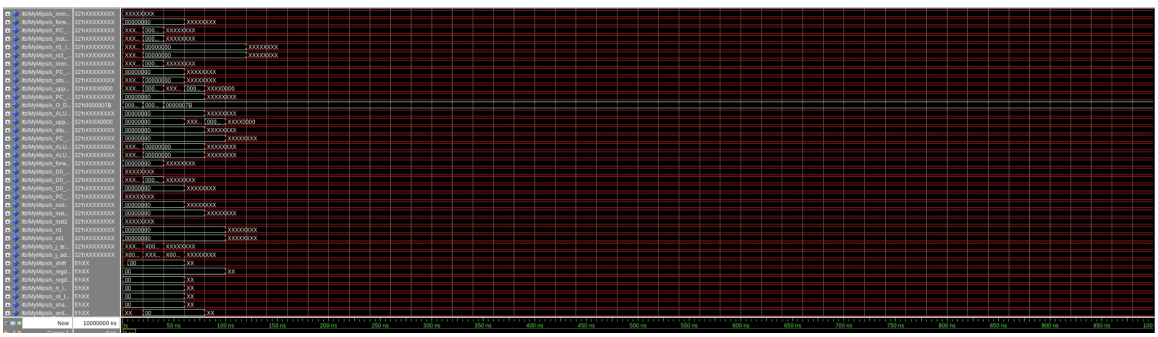
This program is supposed to avoid all control and data hazards, but unfortunately we did continuous testing and could not figure out why the memory read/write didn't work in our processor.

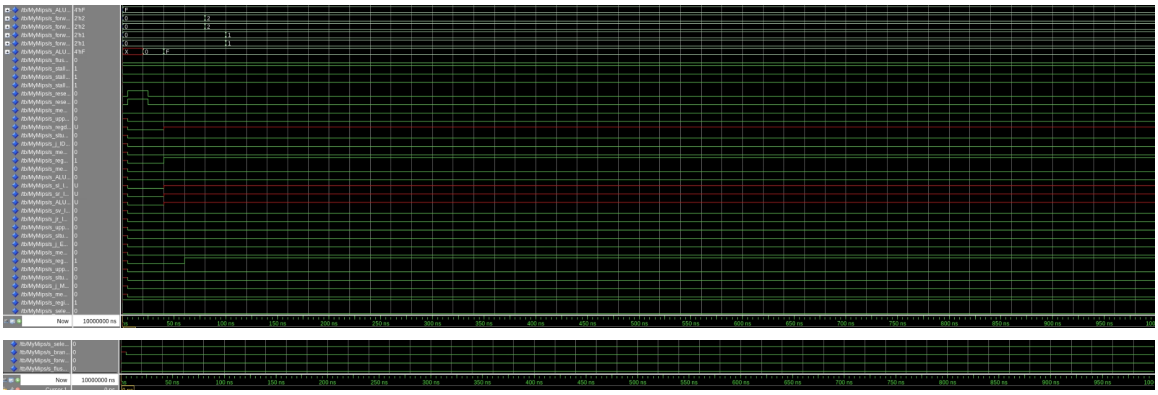
[1.c.ii] Include an annotated waveform in your writeup of two iterations or recursions of these programs executing correctly and provide a short discussion of result correctness. In your waveform and annotation, provide 3 different examples (at least one data-flow and one control-flow) of where you did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

```
Starting Simulation for : mips/bubblesort_2.0.s
starting compilation...
Successfully compiled vhdl
Starting VHDL Simulation...
Successfully simulated program!
Oh no...

Cycle: 5
Incorrect Write to Register File
MARS instruction number: 1      Instruction: lui $1,4097
MARS: Register Write to Reg: 0x01 Val: 0x10010000
Student: Register Write to Reg: 0xXX Val: 0xXXXXXXXX

Cycle: 6
Incorrect Write to Register File
MARS instruction number: 2      Instruction: lui $1,4097
MARS: Register Write to Reg: 0x01 Val: 0x10010000
Student: Register Write to Reg: 0xXX Val: 0xXXXXXXXX

Cycle: 7
Incorrect Write to Register File
MARS instruction number: 3      Instruction: lw $11,28($1)
MARS: Register Write to Reg: 0x0B Val: 0x00000028
Student: Register Write to Reg: 0xXX Val: 0xXXXXXXXX

You have reached the maximum mismatches (3)

Helpful resources for Debugging:
temp/modelsim_dump.txt : output from the VHDL testbench during program execution on your processor
temp/mars_dump.txt : output from MARS containing expected output
vsim.wlf: waveform file generated by processor simulation, you can display this simulation in ModelSim without resimulating yo
ur processor by hand



bash-4.2$
```









for bubblesort_2.0.s it should have done a couple iterations of sorting while being implemented in the pipeline with several NOPS, but unfortunately our processor didn't end up working the way it was supposed to and ended up not reading/writing in the end. I did not end up using the max number on NOPs at the end of a loop or near the start of a new one along with near the end of the program due to not needing as many because of no interference with read/writes for testing.

[For teams with >4]

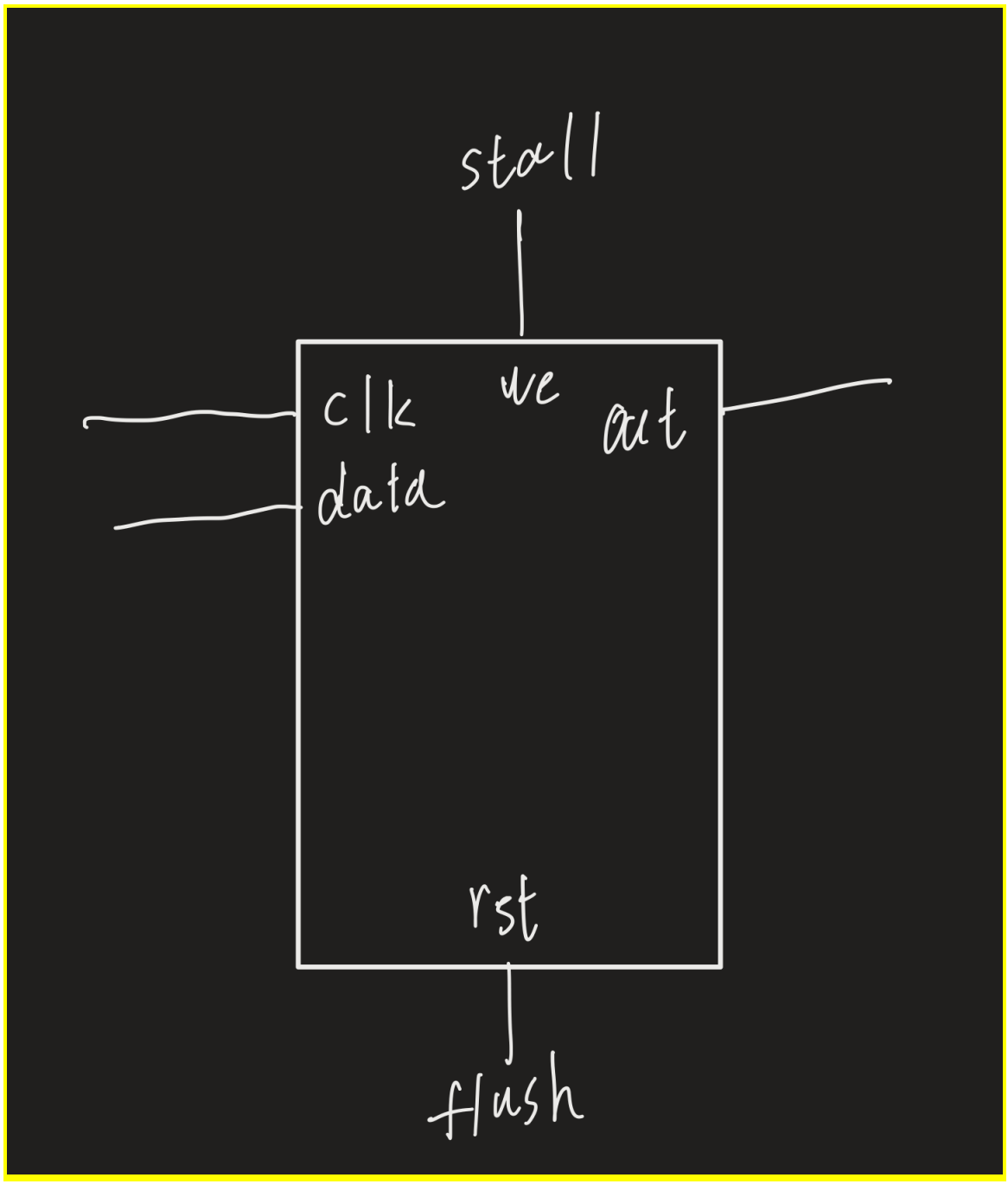
[1.c.iii] Include an annotated waveform in your writeup of two iterations or recursions of these programs executing correctly and provide a short discussion of result correctness. In your waveform and annotation, provide 3 different examples (at least one data-flow and one control-flow) of where you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

We only have two members in the team

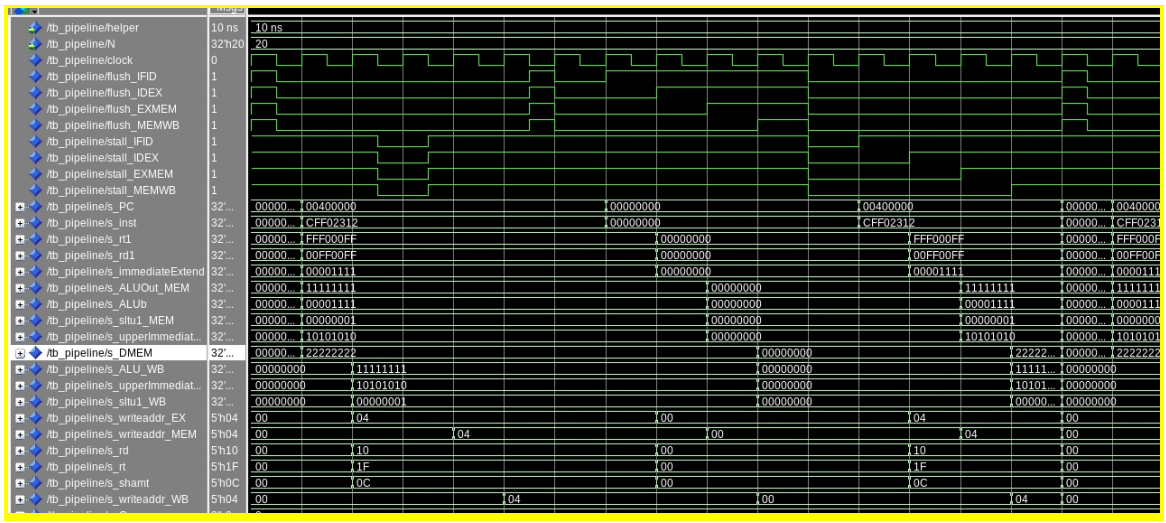
[1.d] report the maximum frequency your software-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).

0 Hz

[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.



testbranch showed that values that are stored in the initial IF/ID stage are available as expected four cycles later. also all four registers can be individually stalled/ flushed

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

Instruction	rt	rd	alu_out	dmem
add	x	x	x	
addi	x		x	
addu	x	x	x	
addiu	x		x	
sub	x	x	x	
subu	x	x	x	
mul	x	x	x	
and	x	x	x	
andi	x		x	
nor	x	x	x	
or	x	x	x	
ori	x		x	
xor	x	x	x	
xori	x		x	
sll	x		x	
sllv	x		x	
srl	x		x	
srlv	x		x	

sra	x		x	
srav	x		x	
slt	x	x	x	
slti	x		x	
sltu	x	x	x	
sltiu	x		x	
lbu	x		x	x
lui	x		x	x
lw	x		x	x

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

Instruction	rt	rd
add	x	x
addi	x	
addu	x	x
addiu	x	
sub	x	x
subu	x	x
mul	x	x
and	x	x
andi	x	
nor	x	x
or	x	x
ori	x	
xor	x	x
xori	x	
sll	x	
sllv	x	x
srl	x	
srlv	x	x
sra	x	
srav	x	x

slt	x	x
sltu	x	x
slti	x	
sltiu	x	
lbu	x	
lui	x	
lw	x	
sw	x	
beq	x	
bne	x	
lb	x	
jr	x	
jal	x	

[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.

Data Dependencies
write ---> read
write ---> write
read ---> write
read ---> read

Stage	Forwarding	Receiving
IF	NO	NO
ID	NO	YES
EX	YES	YES
MEM	YES	YES
WB	YES	NO

require forwarding:
reads a register within 2 cycles after it was written in. (forward MEM → EX if 1 cycle needed, WB → EX if 2 cycles or more needed)
reads a register within 2 cycles after it has been loaded (forward MEM → EX)
require stalls:

reads a register within 1 cycle after it has been loaded

[2.b.iv] global list of the datapath values and control signals that are required during each pipeline stage
Datapath values:
IF: program counter
ID: program counter, instruction(op code, function code), rt_read, rd_read, data_write, immediate
EX: program counter, control_signal, rt_read, rd_read, signed_immediate, immediate
MEM: program counter, control_signal, r_zero, alu_out, data_read, immediate
WB: control_signal, data_read, alu_out, immediate

Control Signals:
IF→ID:s_Inst,s_ALUSrc,s_ALUControl,s_regDst,s_DMemWr,s_branch,s_memtoReg,s_registerWrite_IFID
ID→EX:s_Inst,s_ALUSrc,s_ALUControl,s_regDst,s_DMemWr,s_branch,s_memtoReg,s_registerWrite_IFID
EX→MEM:s_DMemWr,s_branch,s_memtoReg,s_registerWrite_IFID
MEM→WB:s_memtoReg,s_registerWrite_IFID

[2.c.i] list all instructions that may result in a non-sequential PC update and in which pipeline stage that update occurs.
beq:MEM
bne:MEM
j:MEM
jal:MEM
jr:MEM

[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.
IF→ID,ID→EX,EX→MEM stages needs to be flushed for j/jal/jr
IF→ID,ID→EX,EX→MEM stages needs to be stalled for beq/bne if those instructions are not taken(not branching)
IF→ID,ID→EX,EX→MEM stages needs to be flushed for beq/bne if those instructions are taken(branching)

[For teams of 5]
[2.c.iii] As a start, identify and justify the maximum possible benefit per control-flow instruction you will see over your baseline design.
We only have two members in the team

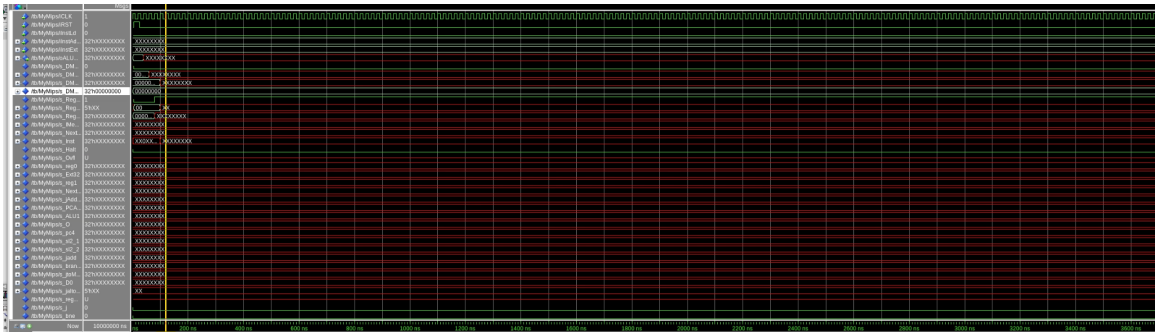
[For teams of 6]
[2.c.iv] As a start, identify and justify the maximum possible benefit per branch you will see over your baseline design.
We only have two members in the team

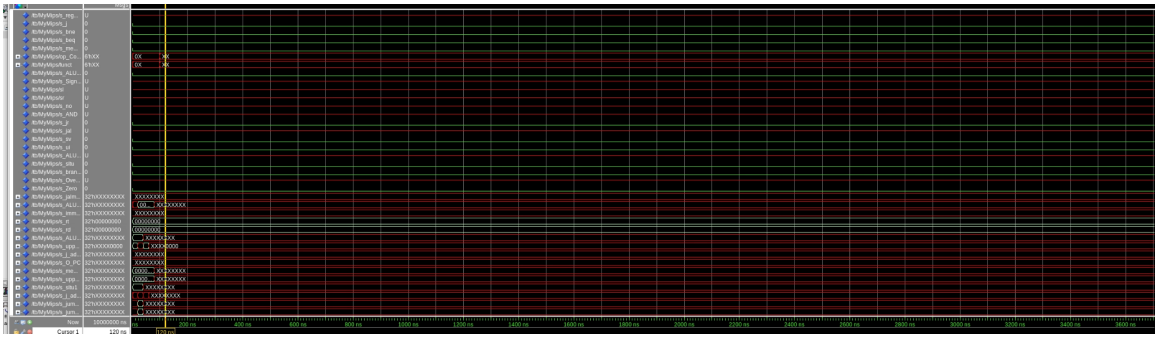
[2.d] implement the hardware-scheduled pipeline using only structural VHDL. As with the previous processors that you have implemented, start with a high-level schematic drawing of the interconnection between components.

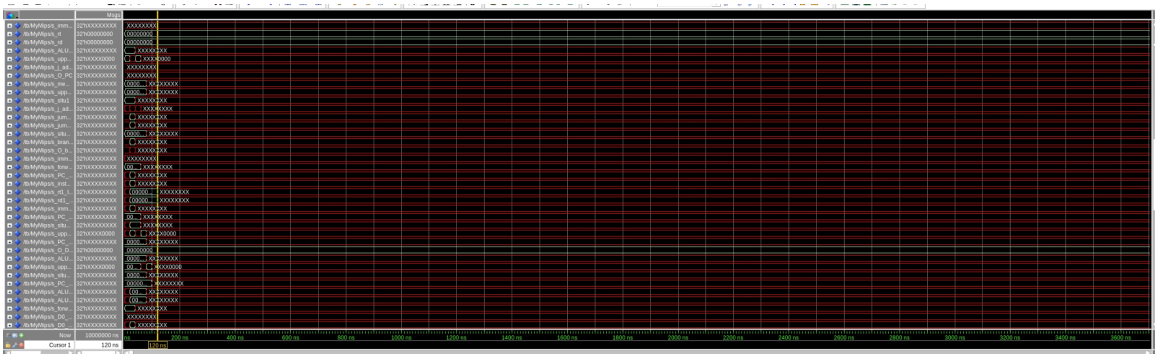
Implementation has been completed

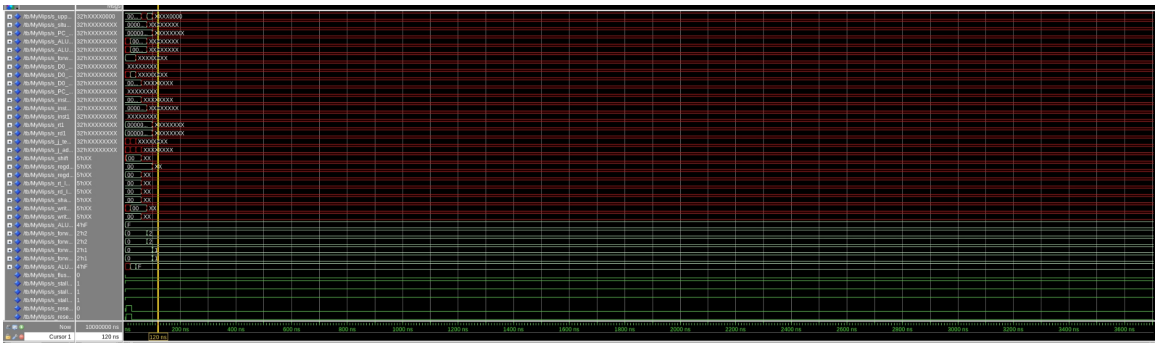
[2.e – i, ii, and iii] In your writeup, show the Modelsim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

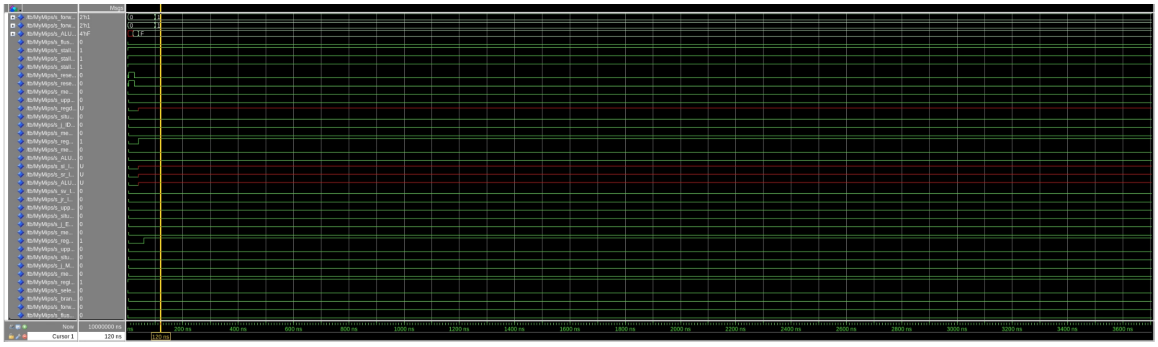
```
Starting Simulation for : mips/fibonacci.s
starting compilation...
Successfully compiled vhdl
Starting VHDL Simulation...
Successfully simulated program!
Oh no...

Cycle: 5
Incorrect Write to Register File
MARS instruction number: 1      Instruction: lui $1,4097
MARS: Register Write to Reg: 0x01 Val: 0x10010000
Student: Register Write to Reg: 0xXX Val: 0xXXXXXXXX

Cycle: 6
Incorrect Write to Register File
MARS instruction number: 2      Instruction: ori $16,$1,0
MARS: Register Write to Reg: 0x10 Val: 0x10010000
Student: Register Write to Reg: 0xXX Val: 0xXXXXXXXX

Cycle: 7
Incorrect Write to Register File
MARS instruction number: 3      Instruction: lui $1,4097
MARS: Register Write to Reg: 0x01 Val: 0x10010000
Student: Register Write to Reg: 0xXX Val: 0xXXXXXXXX

You have reached the maximum mismatches (3)

Helpful resources for Debugging:
```











our processor did not work as intended. By analyzing the output, we observed that the register is not being read/write correctly(addr= 0xXX, val = 0xXXXXXXXX). We noticed there is a problem with the control signal, and memory read/write. However, there is not enough time for us to fix these bugs

[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.
our processor did not work

[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.
our processor did not work

[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).
0 Hz