Cpr E 381: Computer Organization and Assembly-Level Programming

Project Part 2 Report

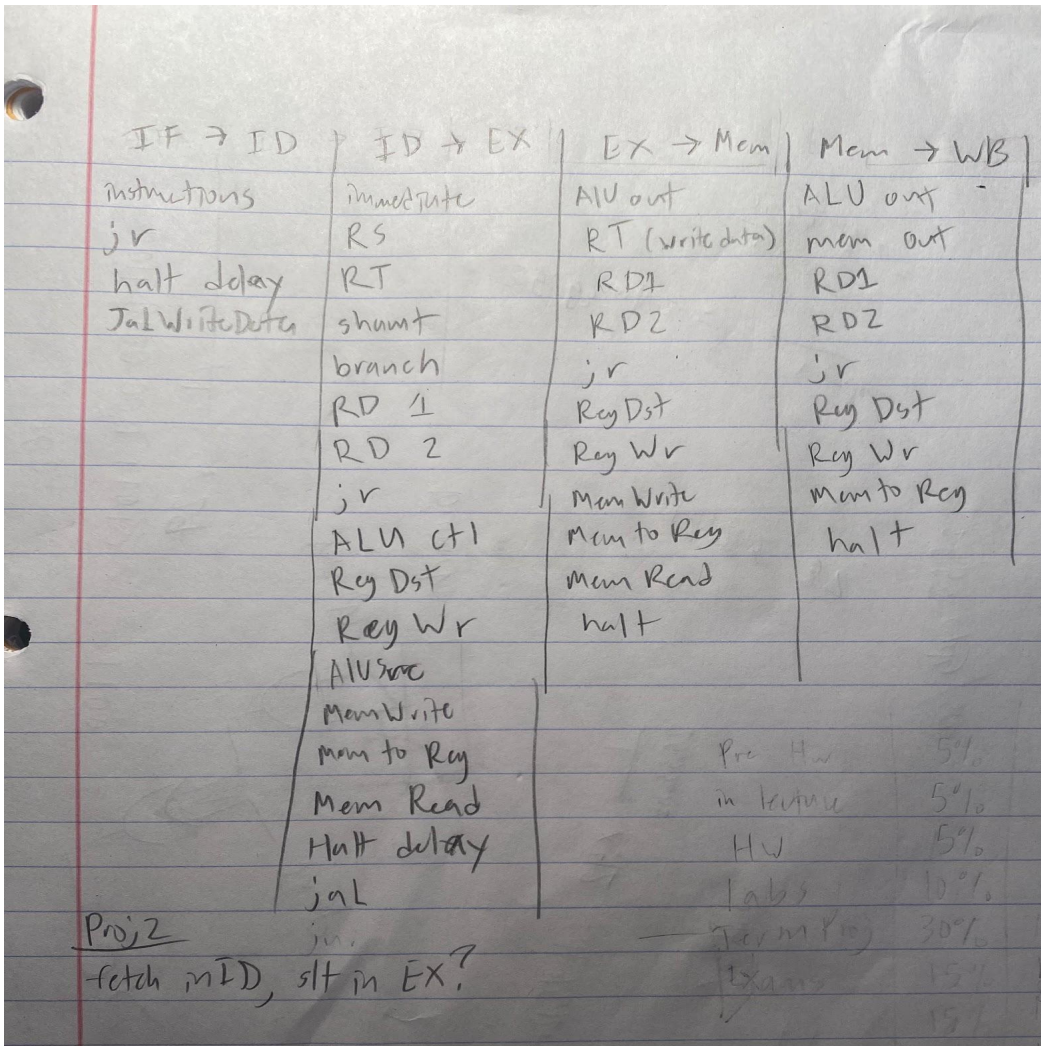
Team Members: __Noah Ross_____

__Alex Brown_____

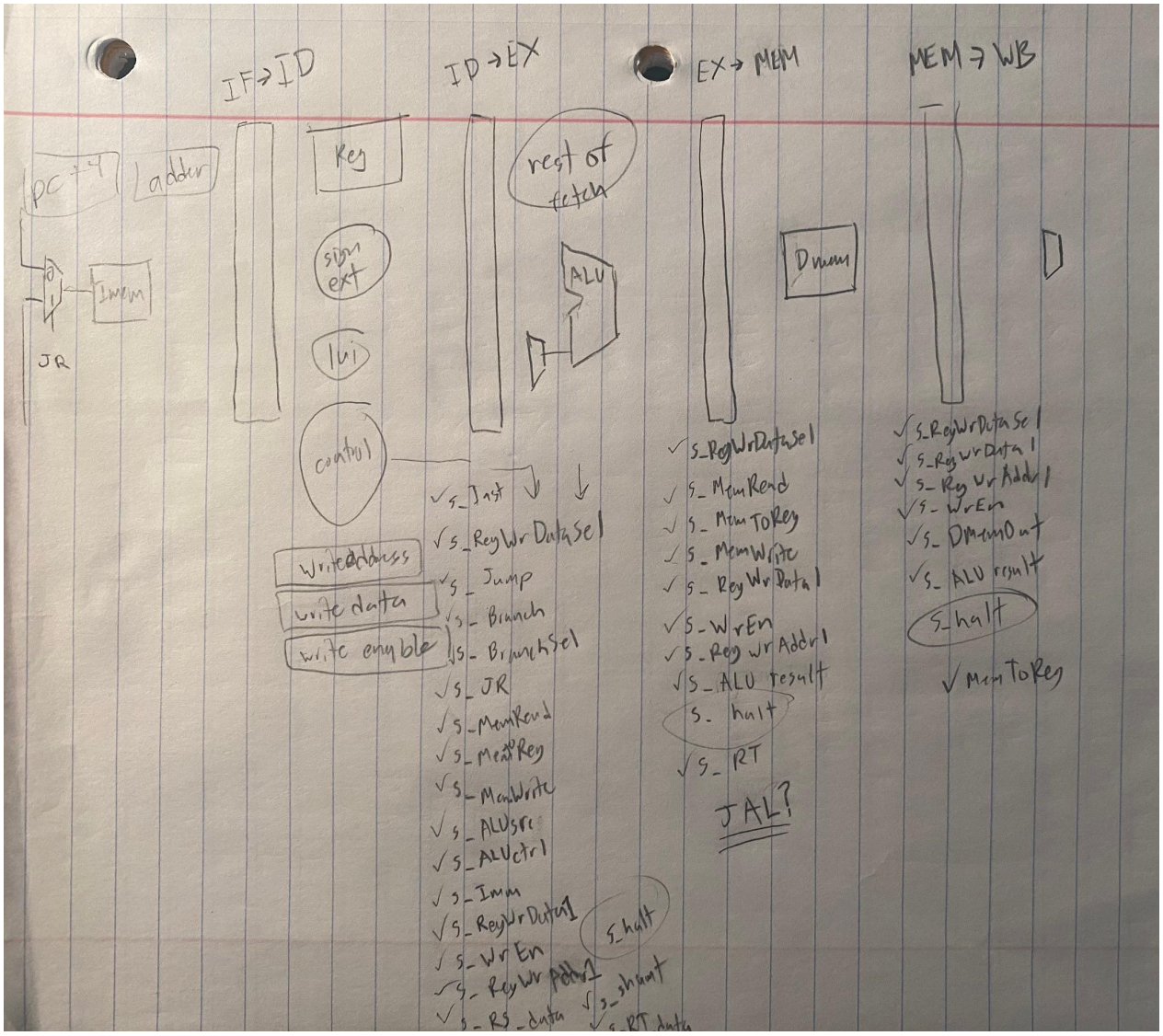
Project Teams Group #:_____5_____

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

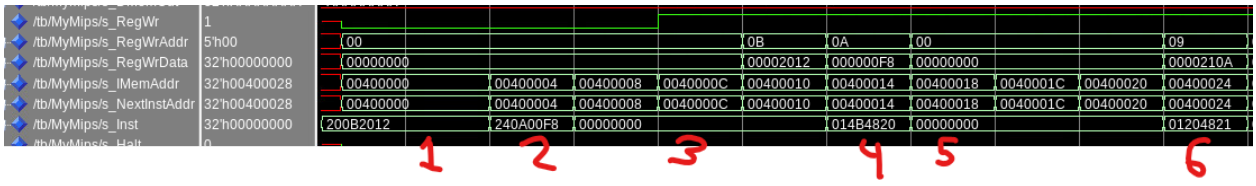
[1.a] Come up with a global list of the datapath values and control signals that are required during each pipeline stage.



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



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

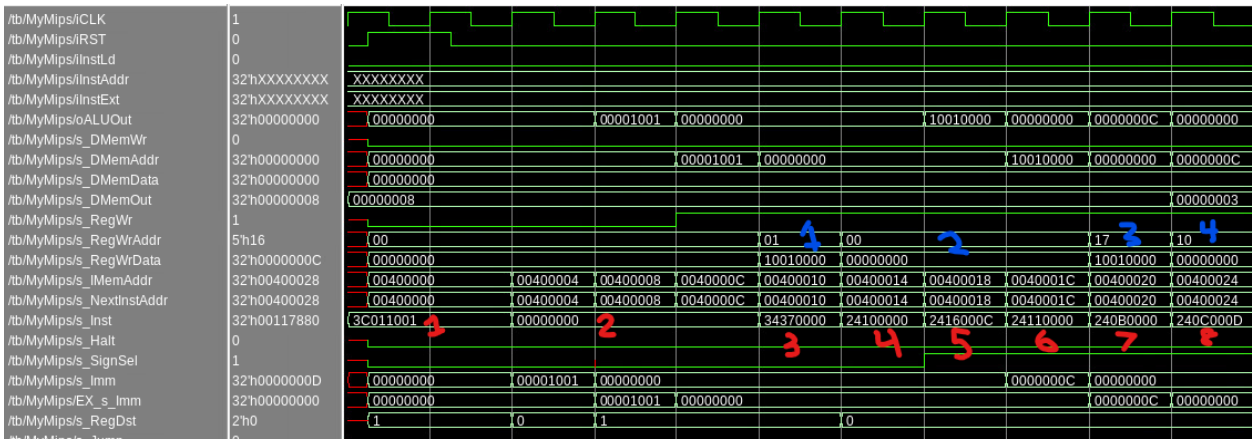


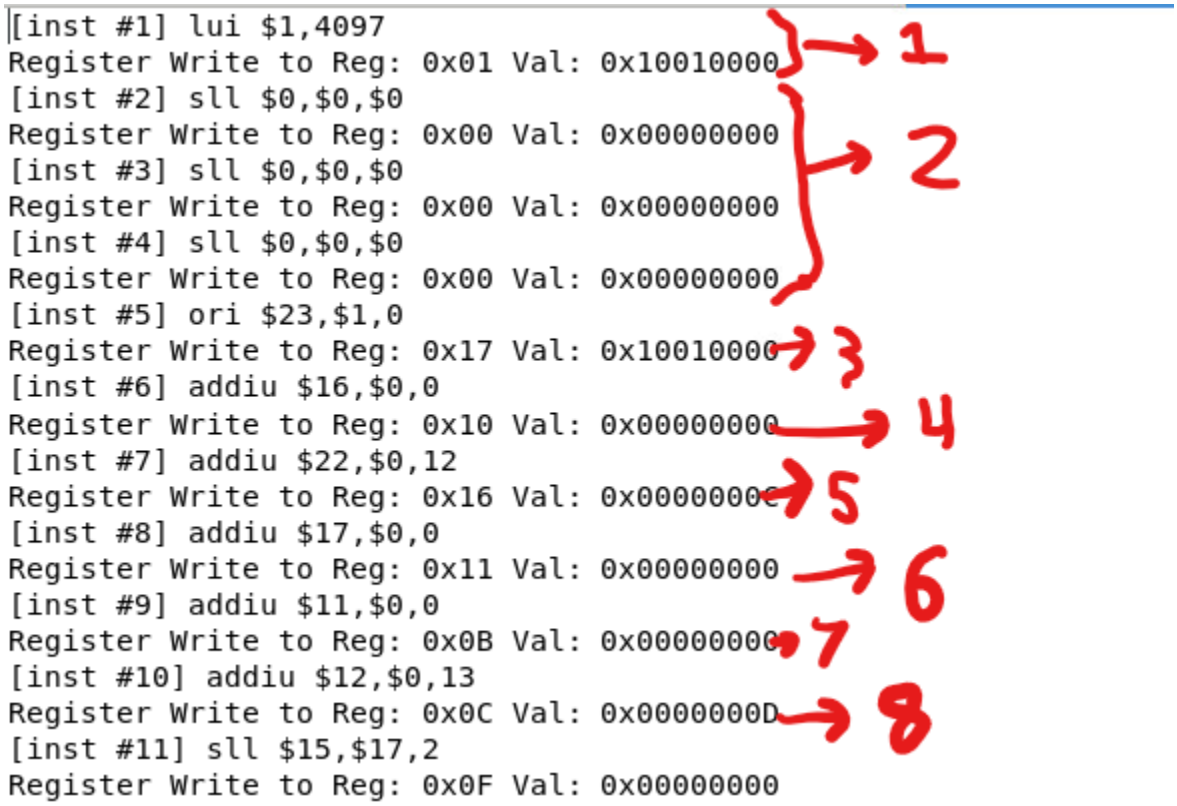

```
addi $t3, $0, 08210   1
addiu $t2, $0, 248    2
nop                   3
nop
nop
add $t1, $t2, $t3     4
nop                   5
nop
nop
addu $t1, $t1, $0     6
```

I checked the entirety of the program by comparing it to what mars.sim had and by checking the Inst and RegWrData to their expected values. Everything matched

[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.

#1



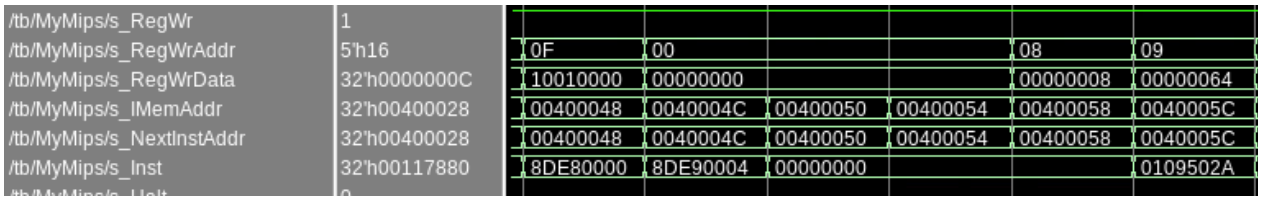


For most of these instructions, no nops were required because the beginning of our bubble sort is just loading the initial values into different registers.

#2

```
lw $t0, 0($t7)                          #load numbers[j]
lw $t1, 4($t7)                          #load numbers[j+1]
nop
nop
nop

slt $t2, $t0, $t1                       #if t0 < t1
```

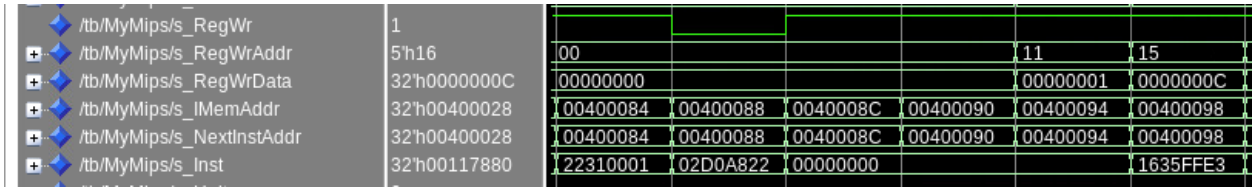


Lw $t0 lw $t1 nop x 3 slt

The lw $t0 instruction requires no nops because the next instruction needs 3, so the instruction is given time to write back.

#3

```
addi $s1, $s1, 1                        #increment t1
sub $s5, $s6, $s0                       #subtract s0 from s6
nop
nop
nop
bne  $s1, $s5, loop                     #if s1 (counter for second loop)
```



Addi sub nop x 3 bne

The addi $s1 instruction requires no nops because the next instruction needs 3, so the instruction is given time to write back.

[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).

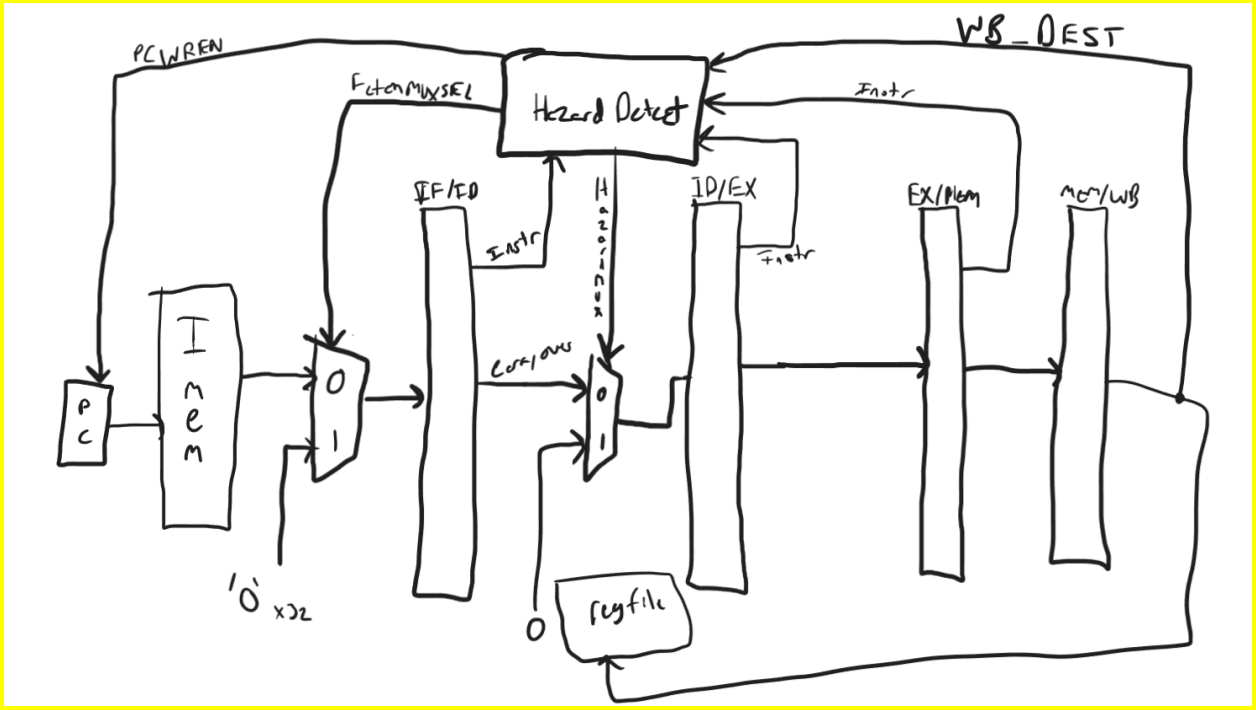
```
FMax: 52.41mhz Clk Constraint: 20.00ns Slack: 0.92ns
```

Our critical path was the alu. The bit shifter and adder and or/ander slow things down greatly and was, therefore, the critical path (the longest cycle). The path goes through a mux for ALU_B input and then the ALU, and then the Fetch Unit.

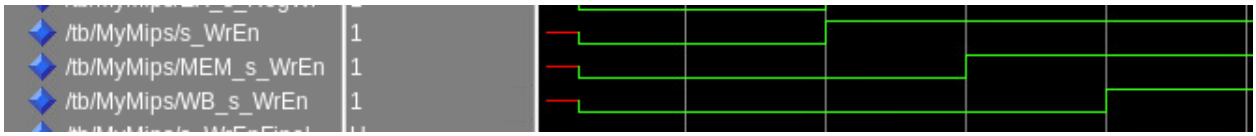
Total (ns)	Incr (ns)		Type	Element
0.000	0.000			launch edge time
3.089	3.089	R		clock network delay
3.321	0.232		uTco	reg_N:RT_Reg|dffg:\G_NBit_REG:0:DFF|s_Q
3.321	0.000	FF	CELL	RT_Reg|\G_NBit_REG:0:DFF|s_Q|q
4.139	0.818	FF	IC	ALU_B_MUX|\G_NBit_MUX:0:MUXI|g_OR2|o_F~0|datad
4.264	0.125	FF	CELL	ALU_B_MUX|\G_NBit_MUX:0:MUXI|g_OR2|o_F~0|combout
4.568	0.304	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:0:ADD|g_AND2|o_F|datab
4.918	0.350	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:0:ADD|g_AND2|o_F|combout
5.217	0.299	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:1:ADD|g_OR|o_F~0|dataa
5.641	0.424	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:1:ADD|g_OR|o_F~0|combout
5.894	0.253	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:2:ADD|g_OR|o_F~0|datad
6.019	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:2:ADD|g_OR|o_F~0|combout
6.267	0.248	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:3:ADD|g_OR|o_F~0|datad
6.392	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:3:ADD|g_OR|o_F~0|combout
6.820	0.428	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:4:ADD|g_OR|o_F~0|datad
6.945	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:4:ADD|g_OR|o_F~0|combout
7.195	0.250	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:5:ADD|g_OR|o_F~0|datad
7.320	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:5:ADD|g_OR|o_F~0|combout
7.571	0.251	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:6:ADD|g_OR|o_F~0|datad
7.696	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:6:ADD|g_OR|o_F~0|combout
7.945	0.249	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:7:ADD|g_OR|o_F~0|datad
8.070	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:7:ADD|g_OR|o_F~0|combout
8.321	0.251	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:8:ADD|g_OR|o_F~0|datad
8.446	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:8:ADD|g_OR|o_F~0|combout
8.705	0.259	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:9:ADD|g_OR|o_F~0|datac
8.986	0.281	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:9:ADD|g_OR|o_F~0|combout
9.244	0.258	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:10:ADD|g_OR|o_F~0|datac
9.525	0.281	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:10:ADD|g_OR|o_F~0|combout
9.780	0.255	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:11:ADD|g_OR|o_F~0|datac
10.061	0.281	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:11:ADD|g_OR|o_F~0|combout
10.314	0.253	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:12:ADD|g_OR|o_F~0|datad
10.439	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:12:ADD|g_OR|o_F~0|combout
10.689	0.250	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:13:ADD|g_OR|o_F~0|datad
10.814	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:13:ADD|g_OR|o_F~0|combout
11.207	0.393	FF	IC	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:14:ADD|g_OR|o_F~0|datad
11.332	0.125	FF	CELL	ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:14:ADD|g_OR|o_F~0|combout

```
    13.366      0.125 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:19:ADD|g_OR|o_F~0|combout
    13.618      0.252 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:20:ADD|g_OR|o_F~0|datad
    13.743      0.125 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:20:ADD|g_OR|o_F~0|combout
    13.991      0.248 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:21:ADD|g_OR|o_F~0|datad
    14.116      0.125 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:21:ADD|g_OR|o_F~0|combout
    14.374      0.258 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:22:ADD|g_OR|o_F~0|datac
    14.655      0.281 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:22:ADD|g_OR|o_F~0|combout
    14.911      0.256 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:23:ADD|g_OR|o_F~0|datac
    15.192      0.281 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:23:ADD|g_OR|o_F~0|combout
    15.443      0.251 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:24:ADD|g_OR|o_F~0|datad
    15.568      0.125 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:24:ADD|g_OR|o_F~0|combout
    15.823      0.255 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:25:ADD|g_OR|o_F~0|datac
    16.104      0.281 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:25:ADD|g_OR|o_F~0|combout
    16.360      0.256 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:26:ADD|g_OR|o_F~0|datac
    16.641      0.281 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:26:ADD|g_OR|o_F~0|combout
    16.892      0.251 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:27:ADD|g_OR|o_F~0|datad
    17.017      0.125 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:27:ADD|g_OR|o_F~0|combout
    17.266      0.249 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:28:ADD|g_OR|o_F~0|datad
    17.391      0.125 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:28:ADD|g_OR|o_F~0|combout
    17.642      0.251 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:29:ADD|g_OR|o_F~0|datad
    17.767      0.125 FF  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:29:ADD|g_OR|o_F~0|combout
    18.154      0.387 FF    IC  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~6|datad
    18.304      0.150 FR  CELL  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~6|combout
    18.506      0.202 RR    IC  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~18|datad
    18.661      0.155 RR  CELL  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~18|combout
    18.866      0.205 RR    IC  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~19|datad
    19.005      0.139 RF  CELL  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~19|combout
    19.254      0.249 FF    IC  EX_Fetch_Unit|branchSel|o_F~0|datad
    19.404      0.150 FR  CELL  EX_Fetch_Unit|branchSel|o_F~0|combout
    19.677      0.273 RR    IC  Fetch_Unit_IF|g_pcreg|\G_NBit_REG:9:REGI|s_Q~2|datac
    19.944      0.267 RF  CELL  Fetch_Unit_IF|g_pcreg|\G_NBit_REG:9:REGI|s_Q~2|combout
    20.813      0.869 FF    IC  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:20:MUXI|g_OR2|o_F~0|datac
    21.094      0.281 FF  CELL  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:20:MUXI|g_OR2|o_F~0|combout
    21.321      0.227 FF    IC  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:20:MUXI|g_OR2|o_F~1|datad
    21.446      0.125 FF  CELL  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:20:MUXI|g_OR2|o_F~1|combout
    21.681      0.235 FF    IC  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:20:MUXI|g_OR2|o_F~2|datac
    21.962      0.281 FF  CELL  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:20:MUXI|g_OR2|o_F~2|combout
    21.962      0.000 FF    IC  Fetch_Unit_IF|g_pcreg|\G_NBit_REG:20:REGI|s_Q|d
    22.066      0.104 FF  CELL  fetchIF:Fetch_Unit_IF|PCreg:g_pcreg|PCdffg:\G_NBit_REG:20:REGI|s_Q
Data Required Path:
Total (ns)  Incr (ns)     Type  Element
```

[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.



Our team didn't make a register for each stage; we made a register for each signal. This was incredibly time-consuming, and I would not do it this way next time, but we were learning what signals we needed one at a time, and that was one way to make it functional. For this reason, we don't have a test bench that can test 4 registers for their ability to carry data to the next clock cycle, but we can show you the cascading signals. The example is Write Enable: you can see it is 1 when it is created in EX, then 1 in the next cycle (MEM), then 1 in the cycle after that (WB). This way, we could physically see each signal where it was at and add and subtract signals with ease.

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

Specifying Signals Specific to the instructions production as opposed to all the other signals that are at play, such as ones that lead aid in consumption.

- **Addi** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData,EX_s_Imm
- **Addiu** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData,EX_s_Imm
- **Andi** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData,EX_s_Imm
- **Lui** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData, EX_s_Imm, s_LUI
- **Lw** - WB_s_WrEn, WB_s_MemtoReg(set to 0) ,WB_s_RegWrData, WB_s_DmemOut
- **Slti** -WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData, EX_s_Imm
- **Sw (value produced in dmem)** - MEM_s_MemWrite, MEM_s_RT, MEM_s_ALUresult
- **Xori** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData, EX_s_Imm
- **Ori** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData, EX_s_Imm
- **Addu** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Sub** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Subu** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **And** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Or** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Xor** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Nor** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Slt** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Add** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **sll** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Srl** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Sra** - WB_s_WrEn, WB_s_MemtoReg(set to 0), WB_s_ALUresult,WB_s_RegWrData
- **Jal** - WB_s_WrEn, WB_s_MemtoReg(set to 0),WB_s_RegWrData,EX_s_PCp4
- **Bgezal** - WB_s_WrEn, WB_s_MemtoReg(set to 0),WB_s_RegWrData,EX_s_PCp4
- **bgtzal** - WB_s_WrEn, WB_s_MemtoReg(set to 0),WB_s_RegWrData,EX_s_PCp4

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

Specifying Signals Specific to the instruction's consumption as opposed to all the other signals that are at play such as ones that lead aid in production.

- **Addi** - ID_s_instruction, EX_s_RS
- **Addiu** - ID_s_instruction, EX_s_RS
- **Andi** - ID_s_instruction, EX_s_RS
- **Lw (consumes from dmem)** - EX_s_RS
- **Sw** - ID_s_instruction, MEM_s_MemWrite
- **Beq** - ID_s_instruction, ID_s_PCp4
- **Bne** - ID_s_instruction, ID_s_PCp4
- **Xori** - ID_s_instruction, EX_s_RS
- **Ori** - ID_s_instruction, EX_s_RS
- **Bgez** - ID_s_instruction, EX_s_RT, EX_s_RS, ID_s_PCp4
- **Bgezal** - ID_s_instruction, EX_s_RT, EX_s_RS, ID_s_PCp4
- **Bgtz** - ID_s_instruction, EX_s_RT, EX_s_RS, ID_s_PCp4
- **Blez** - ID_s_instruction, EX_s_RT, EX_s_RS, ID_s_PCp4
- **Bltzal** - ID_s_instruction, EX_s_RT, EX_s_RS, ID_s_PCp4
- **Blitz** - ID_s_instruction, EX_s_RT, EX_s_RS, ID_s_PCp4
- **Addu** - ID_s_instruction, EX_s_RT, EX_s_RS
- **Sub** - ID_s_instruction, EX_s_RT, EX_s_RS
- **Subi** - ID_s_instruction, EX_s_RT, EX_s_RS
- **And** - ID_s_instruction, EX_s_RT, EX_s_RS
- **Or** - ID_s_instruction, EX_s_RT, EX_s_RS
- **Xor** - ID_s_instruction, EX_s_RT, EX_s_RS
- **Nor** - ID_s_instruction, EX_s_RT, EX_s_RS
- **Slt** - ID_s_instruction, EX_s_RT, EX_s_RS
- **Add** - ID_s_instruction, EX_s_RT, EX_s_RS
- **sll** - ID_s_instruction, EX_s_RT, EX_s_RS
- **srl** -ID_s_instruction, EX_s_RT, EX_s_RS
- **sra** -ID_s_instruction,EX_s_RT, EX_s_RS

Most of these instructions just use signals pertaining to where their data is to be consumed from. There is obviously a lot more that goes into it, but these are the basics for where the consumption comes from.

[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.

All data hazards we need to handle for this project essentially concern Data Production followed immediately by consumption from the data location that just had a produced value. Forwarding from the dmem out to the EXE stage can allow an instruction to move into exe right behind the producing instruction instead of waiting in the ID stage to let

the instruction before it gets to the WB stage. Forwarding could also allow any ALU instruction's result to be brought back to the ALU earlier than it would be if they had to wait for its writeback.

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

IF	ID	EXE	MEM	WB
s_PC_WR_EN	ID_s_instruction	EX_s_ALUSrc	MEM_s_ALUresult	s_HaltDelay_4
IF_s_PCwriteback	ID_s_PCp4	EX_s_RT	MEM_s_RT	WB_s_WrEn
EX_s_Alt	s_RegDst	EX_s_Imm	MEM_s_MemWrite	WB_s_MemtoReg
s_PCp4	s_Jump	EX_s_RS		WB_s_ALUresult
s_NextInstAddr	s_Branch	s_ALU_B		WB_s_DmemOut
s_IMemAddr	s_BranchSel	EX_s_ALUctrl		s_RD_data
s_IF_MUX_SEL	s_JRCtrl	EX_s_shamt		WB_s_RegWrDataSel
	s_JAL	s_ALUresult		WB_s_RegWrData
	s_MemRead	EX_s_Instruction		
	s_MemToReg	EX_s_PCp4		
	s_MemWrite	EX_s_Jump		
	s_ALUSrc	EX_s_Branch		
	EX_s_RegWrAddr	EX_s_JRctrl		
	MEM_s_RegWrAddr	s_Zero		
	WB_s_RegWrA	s_ALU_slt		

	ddr			
	ID_s_Instruction	EX_s_BranchSel		
	EX_s_Instruction	s_WrEnIn		
	MEM_s_Instruction	EX_s_RegWr		
	s_ID_MUX_SEL	EX_s_RegWrSel		
	s_SignSel	EX_s_Alt_MID		
	s_RegWrData1			
	s_LUI			
	s_RegWrAddr1			
	s_RegWrAddr			
	s_RegWr			
	s_RegWrData			

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

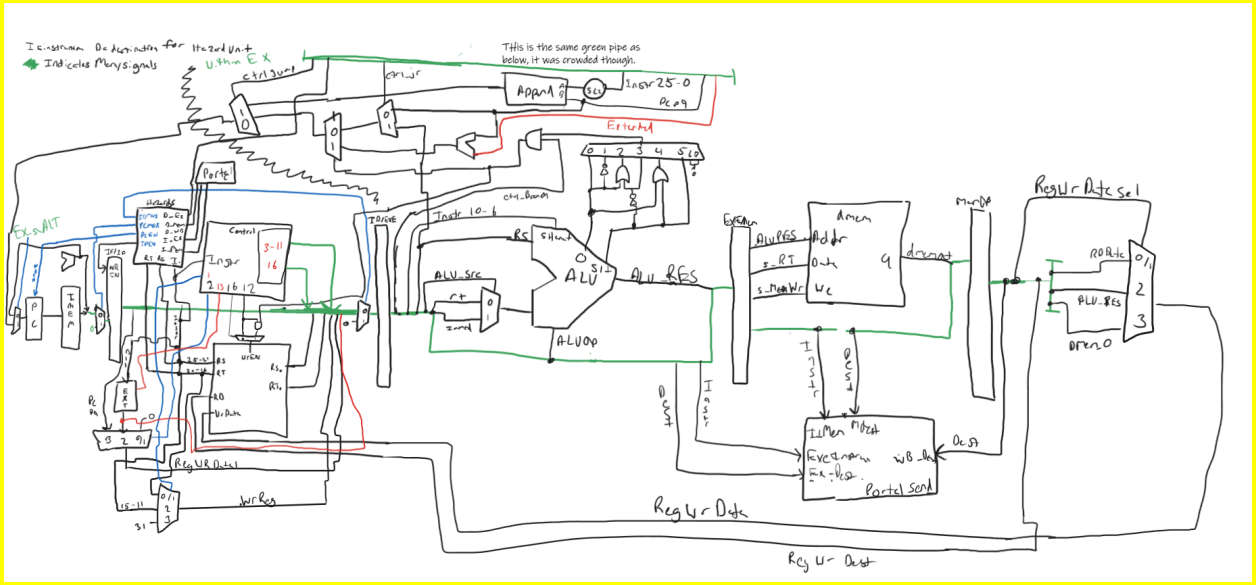
- Beq
- Bne
- Jump
- Jal
- Bgez
- Bgezal
- Bgtz
- Blez
- Bltzal
- Blitz

All of the PC changes occur when these instructions hit the EXE stage of execution.

[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.

When these stages hit the instruction decode, IF and ID need to be stalled, and the IF->ID Register needs to be squashed so that any instructions behind it in IMEM do not make it into the pipeline. Essentially 1 stage needs to be flushed which results in two stages being stalled.

[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.



Control Code:

Number	Output Name	Signal Name
1	o_RegDst	s_RegDst
2	o_RegWrDataSel	s_RegWrDataSel
3	o_Jump	s_Jump
4	o_Branch	s_Branch
5	o_BranchSel	s_BranchSel
6	o_JR	s_JRctrl

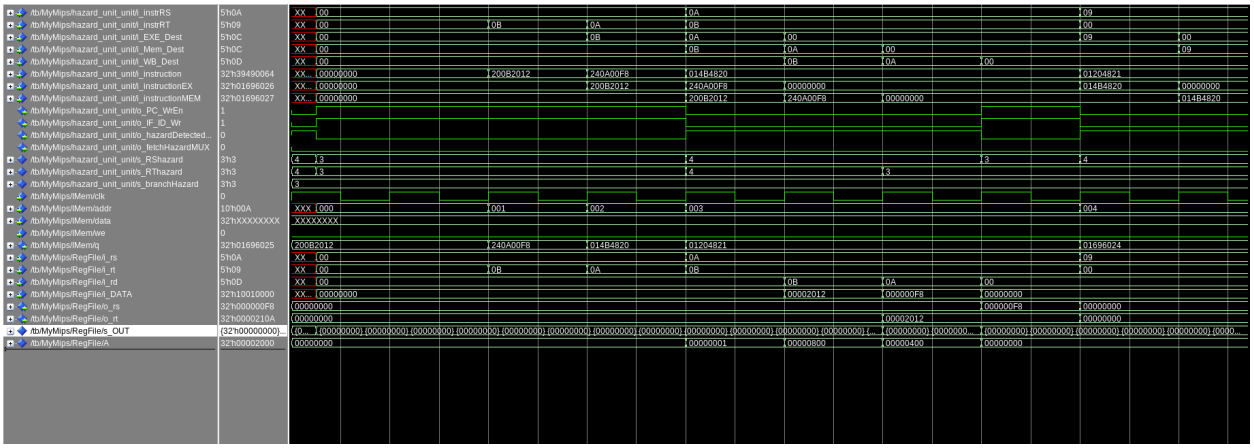
7	o_JAL	s_JAL
8	o_MemRead	s_MemRead
9	o_MemtoReg	s_MemToReg
10	o_MemWrite	s_MemWrite
11	o_ALUSrc	s_ALUSrc
12	o_RegWrite	s_RegWr1
13	o_EXT	s_SignSel
14	o_RegWriteSel	s_RegWrSel
15	o_ALUOp	s_ALUctrl

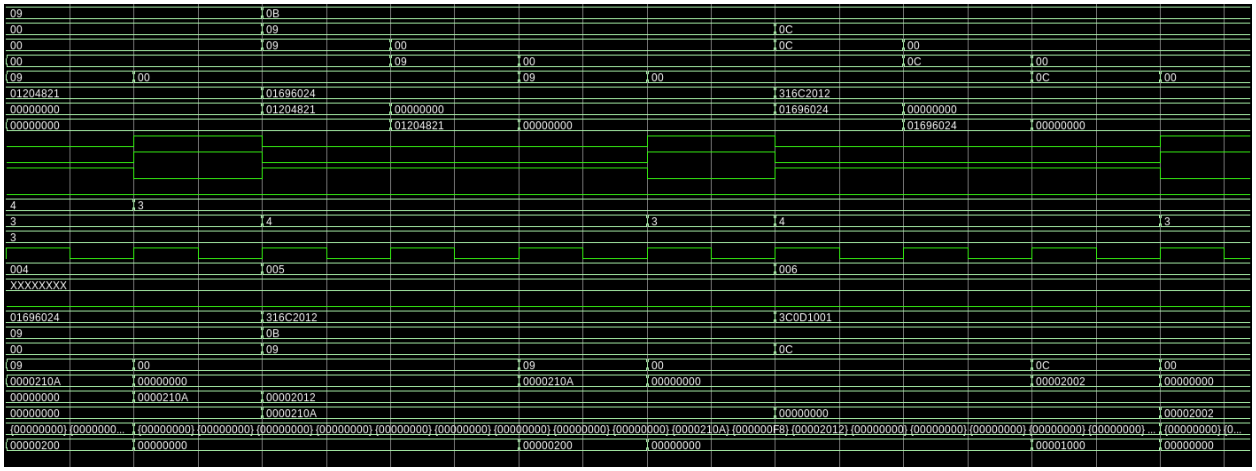
[2.e – i, ii, and iii] In your writeup, show the QuestaSim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

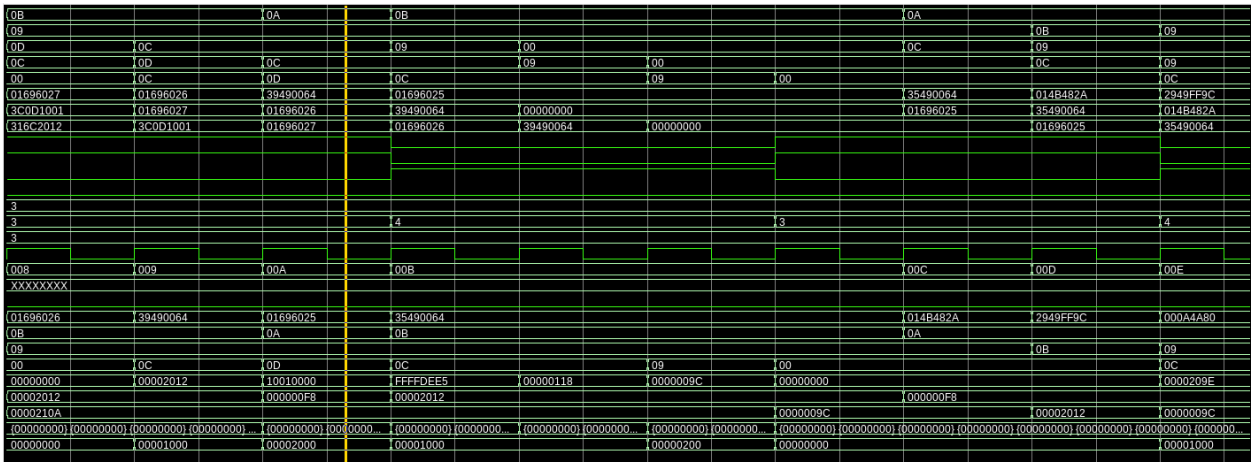
This will be a little difficult to complete as the Hardware Scheduled Pipeline is not fully functional, and most programs cannot be executed properly.

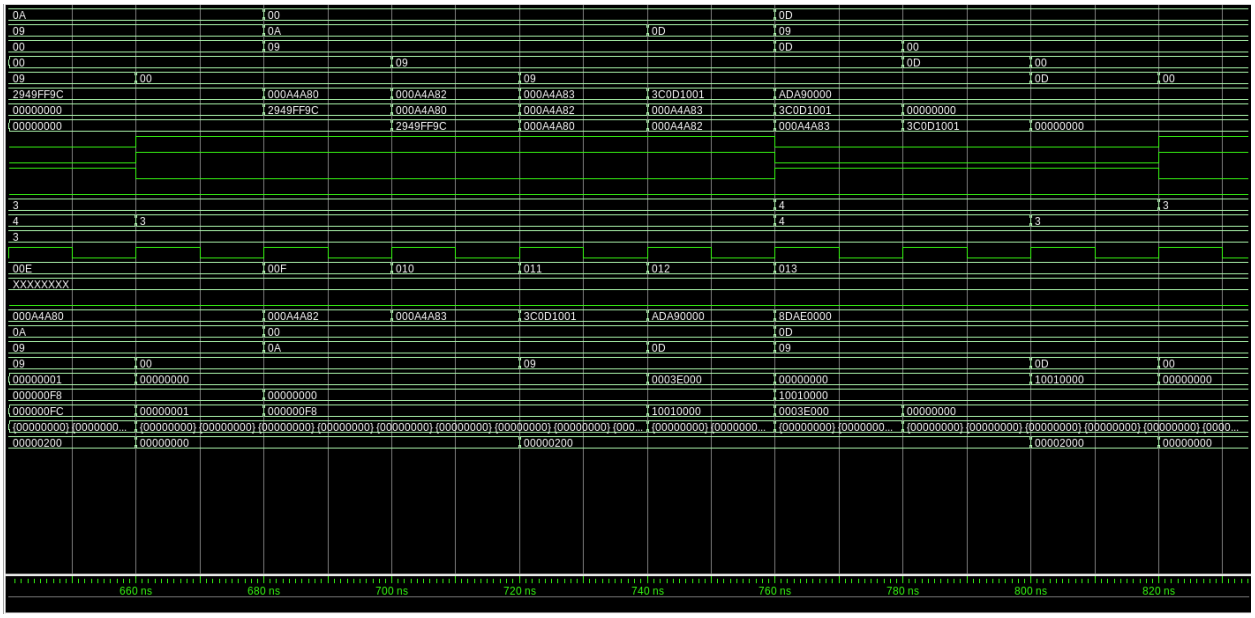
Proj1_Base_Tests.s:

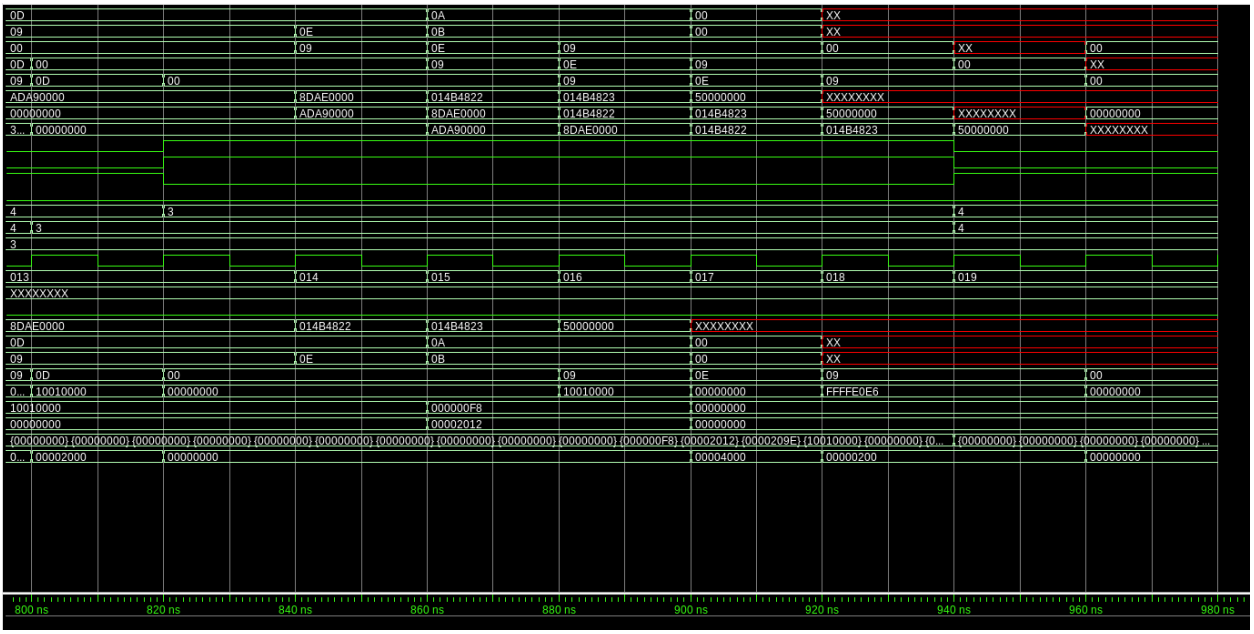
These waveforms show “Hazard_Unit,” “RegFile,” and “Instruction Memory,” as this shows the most important waves visually.


/tb/MyMips/hazard_unit_uniti_instrRS
/tb/MyMips/hazard_unit_uniti_instrRT
/tb/MyMips/hazard_unit_uniti_EXE_Dest
/tb/MyMips/hazard_unit_uniti_Mem_Dest
/tb/MyMips/hazard_unit_uniti_WB_Dest
/tb/MyMips/hazard_unit_uniti_instruction
/tb/MyMips/hazard_unit_uniti_instructionEX
/tb/MyMips/hazard_unit_uniti_instructionMEM
/tb/MyMips/hazard_unit_uniti/o_PC_WrEn
/tb/MyMips/hazard_unit_uniti/o_IF_ID_Wr
/tb/MyMips/hazard_unit_uniti/o_hazardDetected...
/tb/MyMips/hazard_unit_uniti/o_fetchHazardMUX
/tb/MyMips/hazard_unit_uniti/s_RShazard
/tb/MyMips/hazard_unit_uniti/s_RThazard
/tb/MyMips/hazard_unit_uniti/s_branchHazard
/tb/MyMips/IMem/clk
/tb/MyMips/IMem/addr
/tb/MyMips/IMem/data
/tb/MyMips/IMem/we
/tb/MyMips/IMem/q
/tb/MyMips/RegFile/i_rs
/tb/MyMips/RegFile/i_rt
/tb/MyMips/RegFile/i_rd
/tb/MyMips/RegFile/i_DATA
/tb/MyMips/RegFile/o_rs
/tb/MyMips/RegFile/o_rt
/tb/MyMips/RegFile/s_OUT
/tb/MyMips/RegFile/A

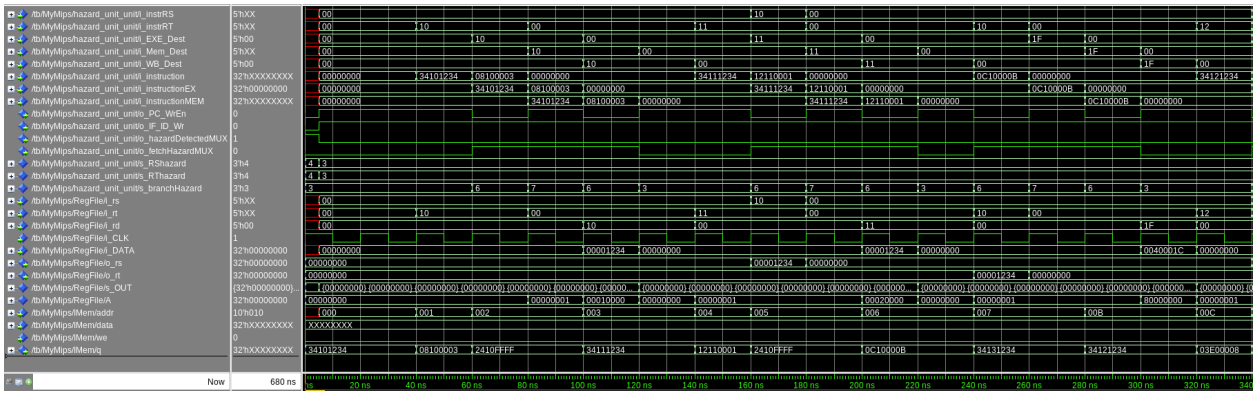

09 0B
00 09 0C
00 09 00 0C 00
00 09 00 0C 00
09 00 09 00 0C 00
01204821 01696024 316C2012
00000000 01204821 00000000 01696024 00000000
00000000 01204821 00000000 01696024 00000000
4 3
3 4 3 4 3
3
004 005 006
XXXXXXXX
01696024 316C2012 3C0D1001
09 0B
00 09 0C
09 00 09 00 0C 00
0000210A 00000000 0000210A 00000000 00002002 00000000
00000000 0000210A 00002012
00000000 0000210A 00000000 00002002
{00000000} {0000000... {00000000} {00000000} {00000000} {00000000} {00000000} {00000000} {00000000} {00000000} {0000210A} {000000F8} {00002012} {00000000} {00000000} {00000000} {00000000} {00000000} ... {00000000} {0...
00000200 00000000 00000200 00000000 00001000 00000000

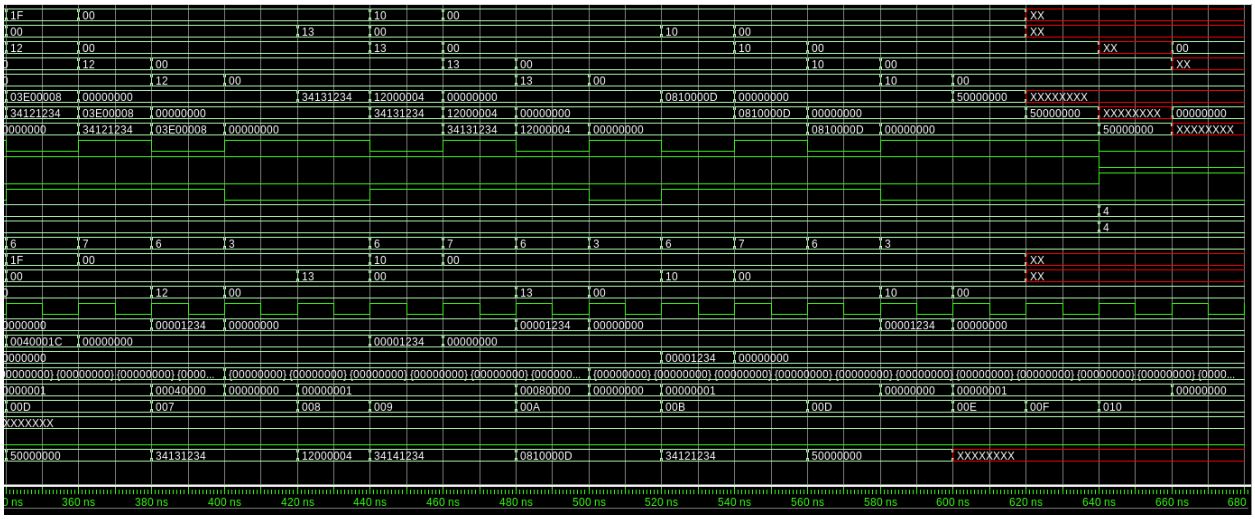

0B 0A 0B 0A
09 0B 09
0D 0C 09 00 0C 09
0C 0D 0C 09 00 0C 09
00 0C 0D 0C 09 00 0C
01696027 01696026 39490064 01696025 35490064 014B482A 2949FF9C
3C0D1001 01696027 01696026 39490064 00000000 01696025 35490064 014B482A
316C2012 3C0D1001 01696027 01696026 39490064 00000000 01696025 35490064
3
3 4 3 4
3
008 009 00A 00B 00C 00D 00E
XXXXXXXX
01696026 39490064 01696025 35490064 014B482A 2949FF9C 000A4A80
0B 0A 0B 0A
09 0B 09
00 0C 0D 0C 09 00 0C
00000000 00002012 10010000 FFFFDEE5 00000118 0000009C 00000000 0000209E
00002012 000000F8 00002012 000000F8
0000210A 0000009C 00002012 0000009C
{00000000} {00000000} {00000000} {00000000} ... {00000000} {00... {00000000} {0000000... {00000000} {0000000... {00000000} {0000000... {00000000} {00000000} {00000000} {00000000} {00000000} {00000000} {00000000} {000000...
00000000 00001000 00002000 00001000 00000200 00000000 00001000


660 ns
680 ns
700 ns
720 ns
740 ns
760 ns
780 ns
800 ns
820 ns


800 ns
820 ns
840 ns
860 ns
880 ns
900 ns
920 ns
940 ns
960 ns
980 ns

simpleBranch.s:





*Simple branch doesn’t complete as desired due to a problem with stalling after a beq instruction.

[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.

WAR hazards should be impossible, given the structure of the pipeline. Forwarding will not be demonstrated as no functionality for it was created. Given the time, we could create a forwarding unit in the EXE stage by pulling data from the cross-stage registers and porting them to beefed-up MUXs before each input of the ALU. This would also create a need for a more involved hazard detection unit to control the forwarding unit and would also re-open the ability to experience WAR hazards in the event that forwarding was not implemented correctly.

Instruction	Hazard Type	Success?

addi $t3, $0, 08210	None	NA
addiu $t2, $0, 248	None	NA
add $t1, $t2, $t3	RAW	YES
addu $t1, $t1, $0	RAW	YES
and $t4, $t3, $t1	RAW	YES
andi $t4, $t3, 08210	None	NA
lui $t5, 0x1001	None	NA
nor $t4, $t3, $t1	None	NA
xor $t4, $t3, $t1	None	NA
xori $t1, $t2, 100	None	NA
or $t4, $t3, $t1	RAW	YES
ori $t1, $t2, 100	None (Could have one if prior didnt stall though)	NA
slt $t1, $t2, $t3	None	NA
slti $t1, $t2, -100	None	NA
sll $t1, $t2, 10	None	NA
srl $t1, $t2, 10	None	NA
sra $t1, $t2, 10	None	NA
lui $t5, 0x1001	None	NA
sw $t1, 0($t5)	RAW	YES
lw $t6, 0($t5)	None (Both cant hit mem at the same time)	NA
sub $t1, $t2, $t3	None	NA
subu $t1, $t2, $t3	None	NA

[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.

Line	Control Hazard Present	Success?
main:	NA	NA
ori $s0, $zero 0x1234	No	NA
j skip	CH	YES
li $s0 0xffffffff	Never Reached	This is correct and desired
skip:	NA	NA
ori $s1 $zero 0x1234	No	NA
beq $s0 $s1 skip2	RAW and CH	YES (for both)
li $s0 0xffffffff	Never Reached	This is correct and desired
skip2:	NA	NA
jal fun	CH	YES
ori $s3 $zero 0x1234	No	NA
beq $s0, $zero exit	CH	FAIL but not on the CH handling. Beq just doesn't work for this instruction and branches despite the condition being false.
ori $s4 $zero 0x1234	Never Reached	We want this to occur.
j exit	CH BUT NEVER REACHED	Would be handled appropriately if it were to be reached.
fun:	NA	NA
ori $s2 $zero 0x1234	No	NA
jr $ra	CH (no RAW because CH in the JAL delayed it enough to have written to $ra)	YES
exit: halt	NA	NA

[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).

```
FMax: 52.84mhz Clk Constraint: 20.00ns Slack: 1.08ns
```

As expected, Our critical path was the alu, similar to the software-enabled pipeline. The bit shifter and adder and or/ander slow things down greatly and were, therefore, the critical path (the longest cycle). The path goes through a mux for ALU_B input and then the ALU, and then the Fetch Unit.

```
Total (ns)  Incr (ns)     Type  Element
==========  ========= ==  ====  ===================================
     0.000      0.000           launch edge time
     3.081      3.081  R        clock network delay
     3.313      0.232     uTco  reg_N:RT_Reg|dffg:\G_NBit_REG:0:DFF|s_Q
     3.313      0.000 FF  CELL  RT_Reg|\G_NBit_REG:0:DFF|s_Q|q
     3.702      0.389 FF    IC  ALU_B_MUX|\G_NBit_MUX:0:MUXI|g_OR2|o_F~0|datab
     4.127      0.425 FF  CELL  ALU_B_MUX|\G_NBit_MUX:0:MUXI|g_OR2|o_F~0|combout
     4.741      0.614 FF    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:0:ADD|g_AND1|o_F|datac
     5.001      0.260 FR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:0:ADD|g_AND1|o_F|combout
     5.261      0.260 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:1:ADD|g_OR|o_F~0|datab
     5.663      0.402 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:1:ADD|g_OR|o_F~0|combout
     5.891      0.228 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:2:ADD|g_OR|o_F~0|datad
     6.046      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:2:ADD|g_OR|o_F~0|combout
     6.273      0.227 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:3:ADD|g_OR|o_F~0|datad
     6.428      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:3:ADD|g_OR|o_F~0|combout
     6.655      0.227 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:4:ADD|g_OR|o_F~0|datad
     6.810      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:4:ADD|g_OR|o_F~0|combout
     7.187      0.377 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:5:ADD|g_OR|o_F~0|datad
     7.342      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:5:ADD|g_OR|o_F~0|combout
     7.569      0.227 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:6:ADD|g_OR|o_F~0|datad
     7.724      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:6:ADD|g_OR|o_F~0|combout
     7.952      0.228 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:7:ADD|g_OR|o_F~0|datad
     8.107      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:7:ADD|g_OR|o_F~0|combout
     8.334      0.227 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:8:ADD|g_OR|o_F~0|datad
     8.489      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:8:ADD|g_OR|o_F~0|combout
     8.716      0.227 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:9:ADD|g_OR|o_F~0|datad
     8.871      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:9:ADD|g_OR|o_F~0|combout
     9.098      0.227 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:10:ADD|g_OR|o_F~0|datad
     9.253      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:10:ADD|g_OR|o_F~0|combout
     9.481      0.228 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:11:ADD|g_OR|o_F~0|datad
     9.636      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:11:ADD|g_OR|o_F~0|combout
     9.866      0.230 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:12:ADD|g_OR|o_F~0|datad
    10.021      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:12:ADD|g_OR|o_F~0|combout
    10.247      0.226 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:13:ADD|g_OR|o_F~0|datad
    10.402      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:13:ADD|g_OR|o_F~0|combout
    10.824      0.422 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:14:ADD|g_OR|o_F~0|datad
    10.979      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:14:ADD|g_OR|o_F~0|combout
    11.207      0.228 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:15:ADD|g_OR|o_F~0|datad
    11.362      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:15:ADD|g_OR|o_F~0|combout
```

```
    12.891      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:19:ADD|g_OR|o_F~0|combout
    13.116      0.225 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:20:ADD|g_OR|o_F~0|datac
    13.403      0.287 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:20:ADD|g_OR|o_F~0|combout
    13.629      0.226 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:21:ADD|g_OR|o_F~0|datac
    13.916      0.287 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:21:ADD|g_OR|o_F~0|combout
    14.141      0.225 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:22:ADD|g_OR|o_F~0|datac
    14.428      0.287 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:22:ADD|g_OR|o_F~0|combout
    14.657      0.229 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:23:ADD|g_OR|o_F~0|datad
    14.812      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:23:ADD|g_OR|o_F~0|combout
    15.036      0.224 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:24:ADD|g_OR|o_F~0|datac
    15.323      0.287 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:24:ADD|g_OR|o_F~0|combout
    15.552      0.229 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:25:ADD|g_OR|o_F~0|datad
    15.707      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:25:ADD|g_OR|o_F~0|combout
    15.935      0.228 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:26:ADD|g_OR|o_F~0|datad
    16.090      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:26:ADD|g_OR|o_F~0|combout
    16.317      0.227 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:27:ADD|g_OR|o_F~0|datad
    16.472      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:27:ADD|g_OR|o_F~0|combout
    16.699      0.227 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:28:ADD|g_OR|o_F~0|datad
    16.854      0.155 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:28:ADD|g_OR|o_F~0|combout
    17.080      0.226 RR    IC  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:29:ADD|g_OR|o_F~0|datac
    17.367      0.287 RR  CELL  ALU_unit|G_ADD_SUB|\G_NBit_ADDSUB:29:ADD|g_OR|o_F~0|combout
    17.777      0.410 RR    IC  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~5|datad
    17.916      0.139 RF  CELL  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~5|combout
    18.143      0.227 FF    IC  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~17|datad
    18.293      0.150 FR  CELL  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~17|combout
    18.498      0.205 RR    IC  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~18|datad
    18.637      0.139 RF  CELL  ALU_unit|G_MUXout|\G_NBit_MUX:31:MUXI|o_F~18|combout
    18.885      0.248 FF    IC  EX_Fetch_Unit|branchSel|o_F~0|datad
    19.035      0.150 FR  CELL  EX_Fetch_Unit|branchSel|o_F~0|combout
    19.302      0.267 RR    IC  Fetch_Unit_IF|g_pcreg|\G_NBit_REG:8:REGI|s_Q~2|datad
    19.441      0.139 RF  CELL  Fetch_Unit_IF|g_pcreg|\G_NBit_REG:8:REGI|s_Q~2|combout
    21.238      1.797 FF    IC  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:5:MUXI|g_OR2|o_F~0|datac
    21.519      0.281 FF  CELL  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:5:MUXI|g_OR2|o_F~0|combout
    21.747      0.228 FF    IC  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:5:MUXI|g_OR2|o_F~1|datad
    21.897      0.150 FR  CELL  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:5:MUXI|g_OR2|o_F~1|combout
    22.102      0.205 RR    IC  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:5:MUXI|g_OR2|o_F~2|datad
    22.257      0.155 RR  CELL  Fetch_Unit_IF|g_MUX|\G_NBit_MUX:5:MUXI|g_OR2|o_F~2|combout
    22.257      0.000 RR    IC  Fetch_Unit_IF|g_pcreg|\G_NBit_REG:5:REGI|s_Q|d
    22.344      0.087 RR  CELL  fetchIF:Fetch_Unit_IF|PCreg:g_pcreg|PCdffg:\G_NBit_REG:5:REGI|s_Q
Data Required Path:
Total (ns)  Incr (ns)     Type  Element
```