CprE 381: Computer Organization and Assembly-Level Programming

Project Part 2 Report

Team Members: Evan Gossling

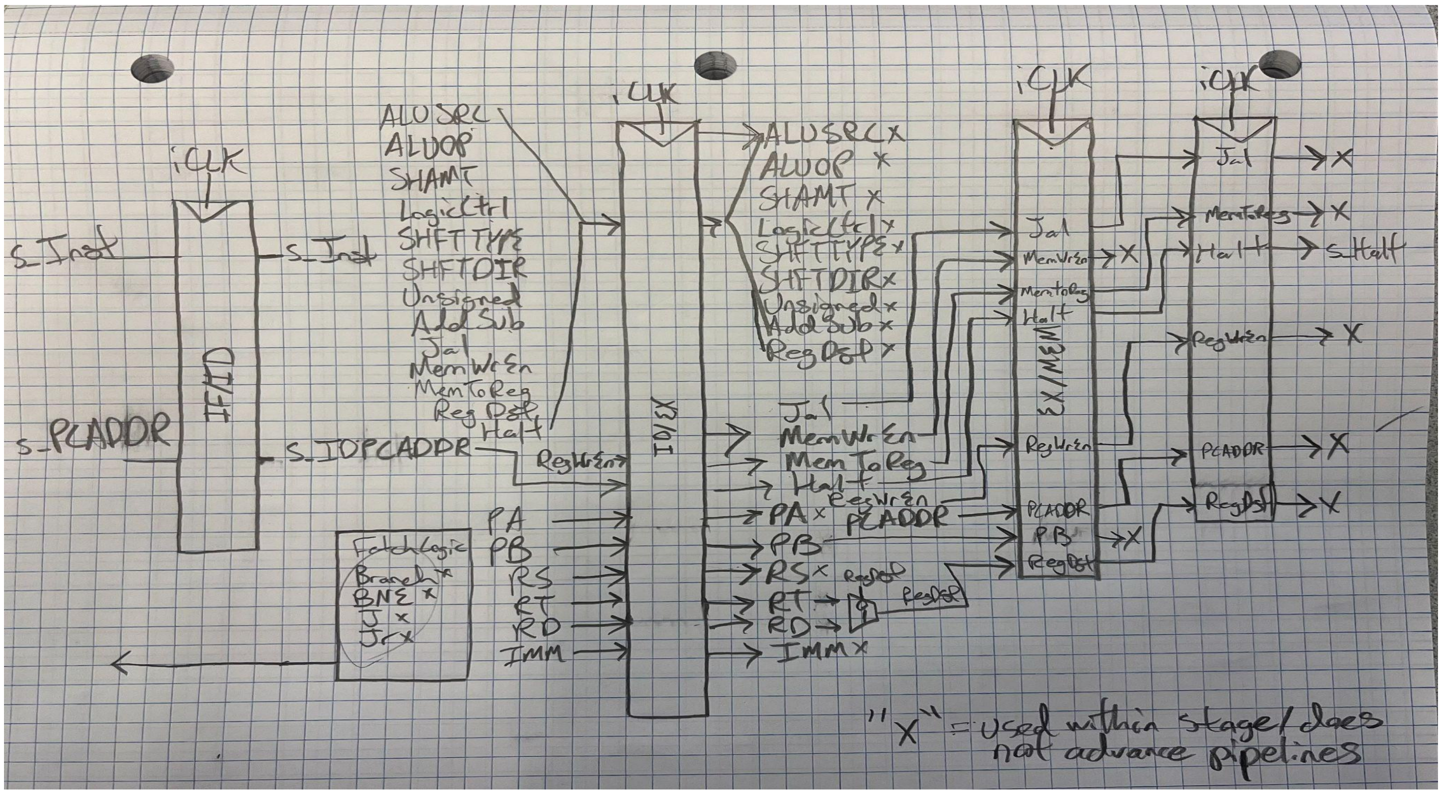
Bridget Schmitt

Jeremy Noesen

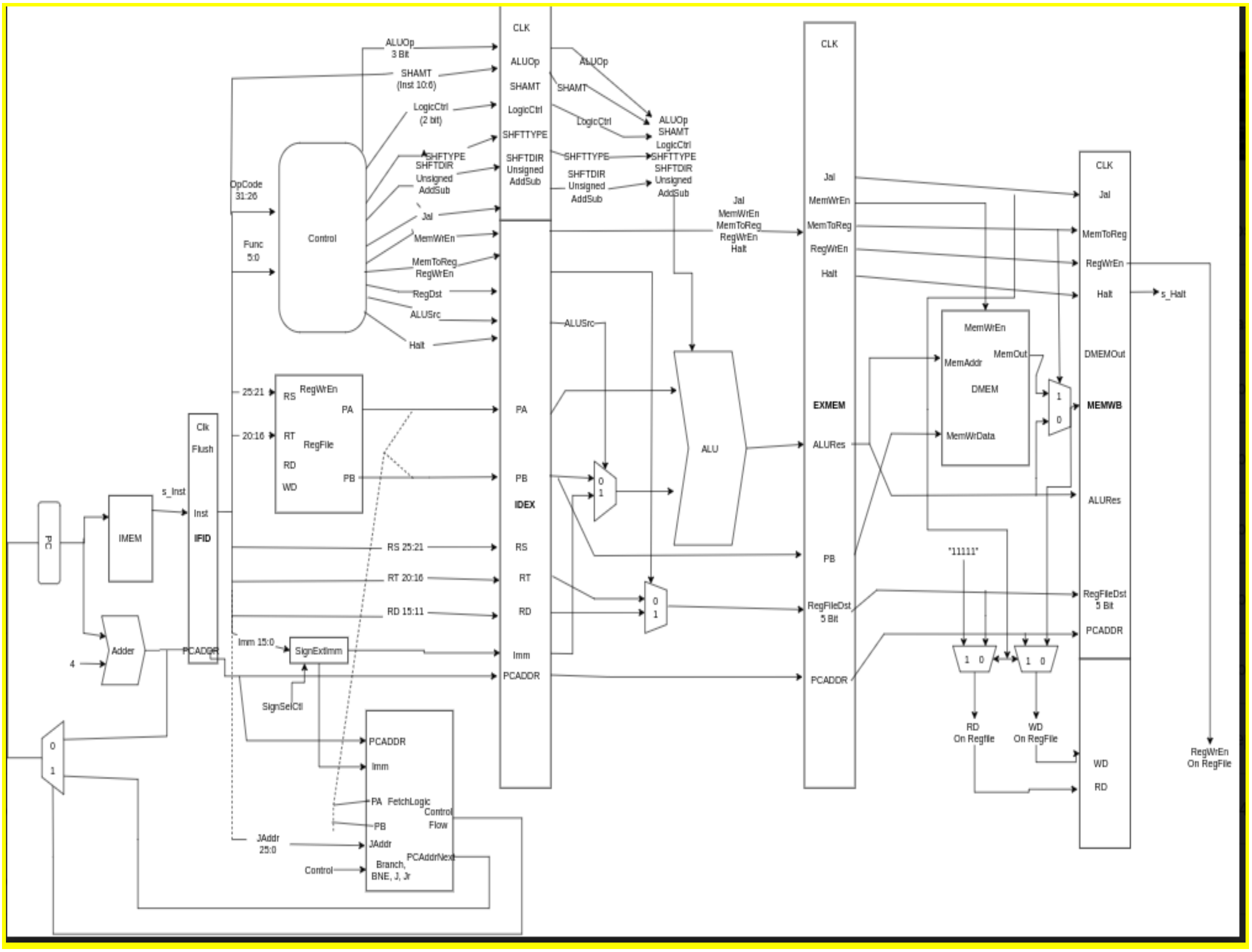
Hunter Northern

Project Teams Group #: Section 5 Group 5

[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.



(Included in report Folder as SoftWareSchem.jpg)

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

The waveform and the Vsim_dumpfile for our software_simple_tests.s file (included as SimpleTestAnnotation.txt, SimpleTestMips.s, and SimpleTestWF.wlf files) we had an arithmetic test from each of the operation types and continued to test with Branches and control flow but the output properly matched the Mars expected outputs as shown in the waveform and vsim_dump (even got a “Victory” message).

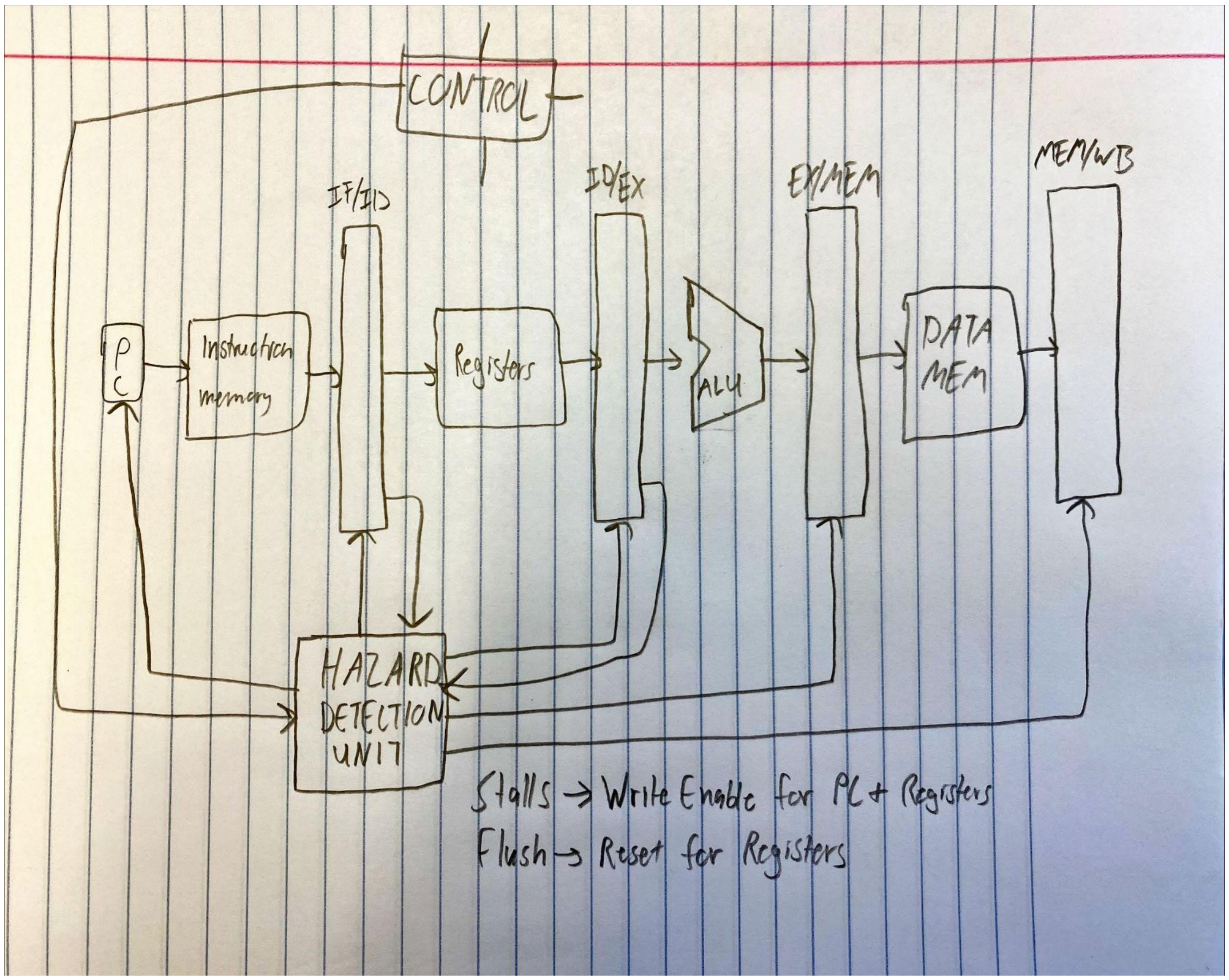
[1.c.ii] Include an annotated waveform in your writeup of two iterations or recursions of these programs executing correctly and provide a short discussion of result correctness. In your waveform and annotation, provide 3 different examples (at least one data-flow and one control-flow) of where you did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

In the Vsim_Dump from the BubbleSortAnnotation.txt file run on the software scheduled processor we ended up getting the expected output as Mars and sorted the array correctly from our checking of each. I have also included the WaveForm as BubbleSortWaveForm which is annotated by each cycle of the Annotation file.

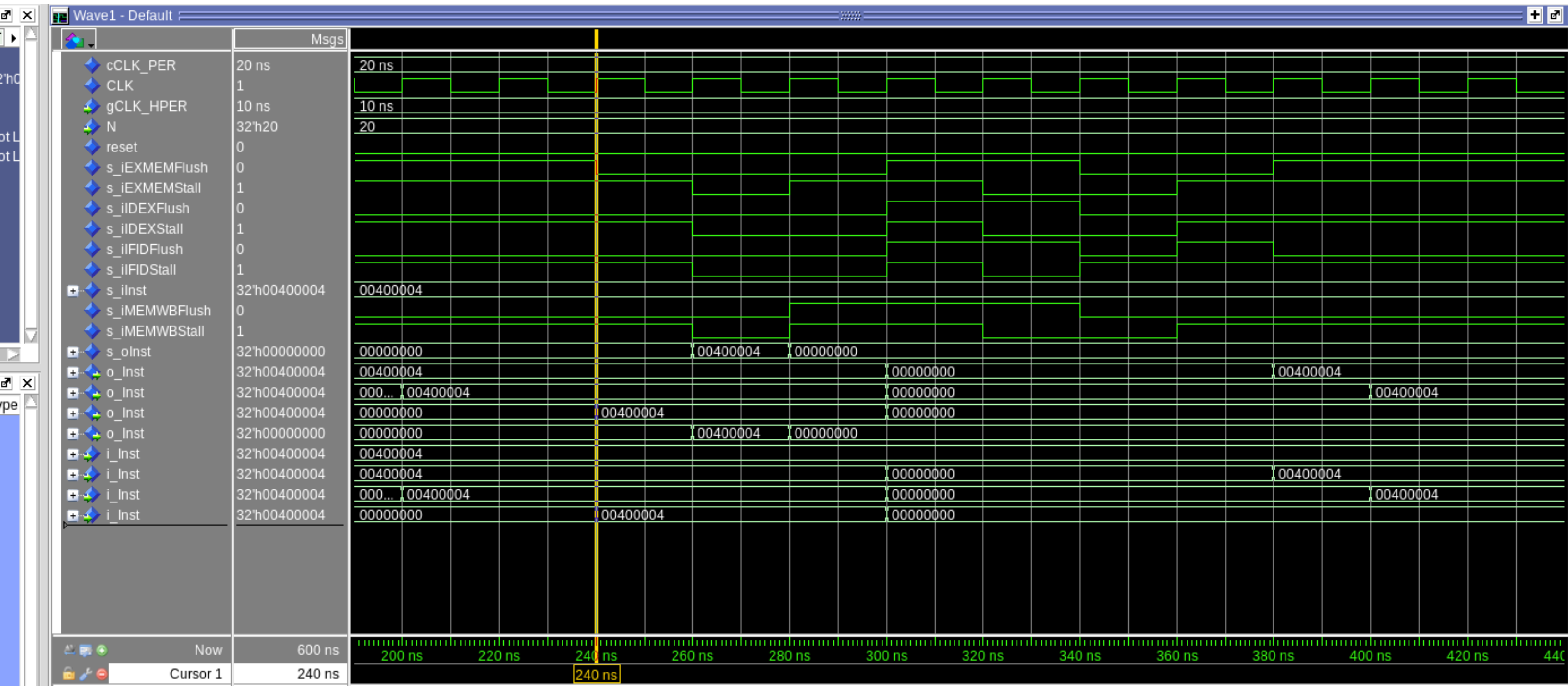
[1.d] report the maximum frequency your software-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).

Full Timing Diagram Given in Folder, FMax : 51.70 mHz with a critical path when going through a branch if equal path as the instruction in one cycle has to go through an adder (subtracts to see if equal) then logic to check zero, then through multiple muxes to check and see if the branch address is being accepted then into another mux to check if we want the PC+4 or a jump address back to the PC Register.

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.



	Msgs
cCLK_PER	20 ns
CLK	1
gCLK_HPER	10 ns
N	32'h20
reset	0
s_iEXMEMFlush	0
s_iEXMEMStall	1
s_iIDEXFlush	0
s_iIDEXStall	1
s_iIFIDFlush	0
s_iIFIDStall	1
s_iInst	32'h00400004
s_iMEMWBFlush	0
s_iMEMWBStall	1
s_oInst	32'h00000000
o_Inst	32'h00400004
o_Inst	32'h00400004
o_Inst	32'h00400004
o_Inst	32'h00000000
i_Inst	32'h00400004
i_Inst	32'h00400004
i_Inst	32'h00400004
i_Inst	32'h00400004

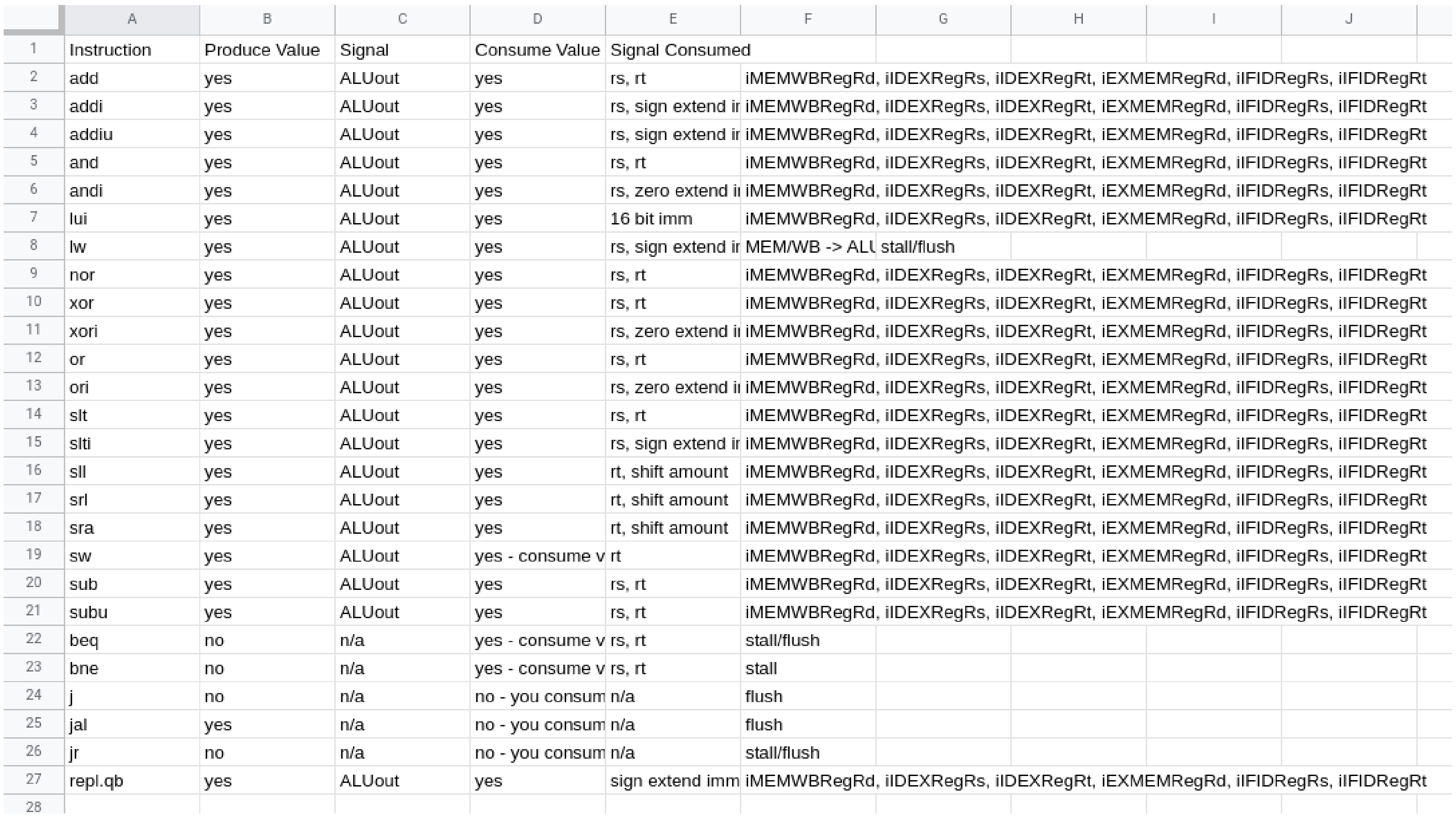
(starts at yellow line and to the right)

Here, I have all of the pipeline registers connected together. I then set the flush and stall of the registers, and I set an instruction to be passed through. The stall is set to the write enable for the registers, so if it is 1 they will write and if they are 0 they will stall. The flush is set to the reset of the registers, so if it is 1, it will reset/flush the registers and if it is 0 they will not be reset/flushed. In the diagram, depending on when everything is flushed or stalled, the output instruction will be changed. Also, the o_Inst is the out instructions from the registers in the pipelined order, same with i_Inst, that is the instruction going into the register in the respective stage. You can see that stalled registers will preserve the value until the next stage and reset values will immediately flush the instruction to 00000000.

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

Shown in table below

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



	A	B	C	D	E	F	G	H	I	J
1	Instruction	Produce Value	Signal	Consume Value	Signal Consumed					
2	add	yes	ALUout	yes	rs, rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
3	addi	yes	ALUout	yes	rs, sign extend i	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
4	addiu	yes	ALUout	yes	rs, sign extend i	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
5	and	yes	ALUout	yes	rs, rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
6	andi	yes	ALUout	yes	rs, zero extend i	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
7	lui	yes	ALUout	yes	16 bit imm	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
8	lw	yes	ALUout	yes	rs, sign extend i	MEM/WB -> AL	stall/flush			
9	nor	yes	ALUout	yes	rs, rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
10	xor	yes	ALUout	yes	rs, rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
11	xori	yes	ALUout	yes	rs, zero extend i	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
12	or	yes	ALUout	yes	rs, rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
13	ori	yes	ALUout	yes	rs, zero extend i	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
14	slt	yes	ALUout	yes	rs, rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
15	slti	yes	ALUout	yes	rs, sign extend i	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
16	sll	yes	ALUout	yes	rt, shift amount	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
17	srl	yes	ALUout	yes	rt, shift amount	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
18	sra	yes	ALUout	yes	rt, shift amount	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
19	sw	yes	ALUout	yes - consume v	rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
20	sub	yes	ALUout	yes	rs, rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
21	subu	yes	ALUout	yes	rs, rt	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
22	beq	no	n/a	yes - consume v	rs, rt	stall/flush				
23	bne	no	n/a	yes - consume v	rs, rt	stall				
24	j	no	n/a	no - you consum	n/a	flush				
25	jal	yes	n/a	no - you consum	n/a	flush				
26	jr	no	n/a	no - you consum	n/a	stall/flush				
27	repl.qb	yes	ALUout	yes	sign extend imm	iMEMWBRegRd, iIDEXRegRs, iIDEXRegRt, iEXMEMRegRd, iIFIDRegRs, iIFIDRegRt				
28										

[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.

DM/WB -> ID/EX(ALU) or

DM/WB -> ID/EX with Stall

ID/EX -> ALU

PC, IF/ID will be stalled and ID/EX will be flushed for load-use

IF/ID will be flushed for jump and PC, IF/ID will be stalled with ID/EX will be flushed for JR

For branch, PC and IF/ID will be stalled and ID/EX will be flushed

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

IF/ID: inputs: clock, Stall, reset, instruction, PC Address

Output: instruction, PC Address

ID/EX: inputs: clock, reset, regA, regB, instruction, RS, RT, RD, immediate, PC Adder, ALUOP, Jal, MemWrEn, MemtoReg, ALUSrc, RegWrEn, RegDst, Add/sub, shift direction, shift type, halt, unsigned select, shift amount, logic Control

Outputs: regA, regB, instruction, RS, RT, RD, immediate, PC Adder, ALUOP, Jal, MemWrEn, MemtoReg, ALUSrc, RegWrEn, RegDst, Add/sub, shift direction, shift type, halt, unsigned select, shift amount, logic Control

EX/MEM: inputs: Clock, Reset, Stall, ALUres, PC Adder, RT, RGDST, jal, MemtoReg, RegWrEn, MemWrEn, Halt, instruction

Output: ALUres, PC Adder, RT, RGDST, jal, MemtoReg, RegWrEn, MemWrEn, Halt, instruction

MEM/WB: inputs: Clock, Reset, Stall, ALIres, PC Adder, MemData, RG destination, Jal, MemtoReg, RegWrEn, Halt, instruction

Outputs: ALUres, PC Adder, MemData, RG destination, Jal, MemtoReg, RegWrEn, Halt

```
  o_ALURES    => s_WBALURES,

  o_PCADDR    => s_WBPCADDR,

  o_MEMDATA  => s_WBMEMDATA,

  o_Jal           => s_WBJal,

  o_MemtoReg => s_WBMemtoReg,

  o_RegWrEn  => s_WBRegWrEn,

  o_Halt         => s_Halt,

  o_RGDST               => s_WBREGDST);
```

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

j, jal, jr, beq, and bne can all result in the PC being changed to a completely different value. These are decided in the IF/ID stage of our processor (we decide which address is next). Non-sequential updates could also occur when a stall happens, the PC value doesn't necessarily update sequentially like it should, it stays on the same value. This is decided in our ID stage with our hazard logic, we send a stall to the PC.

[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.

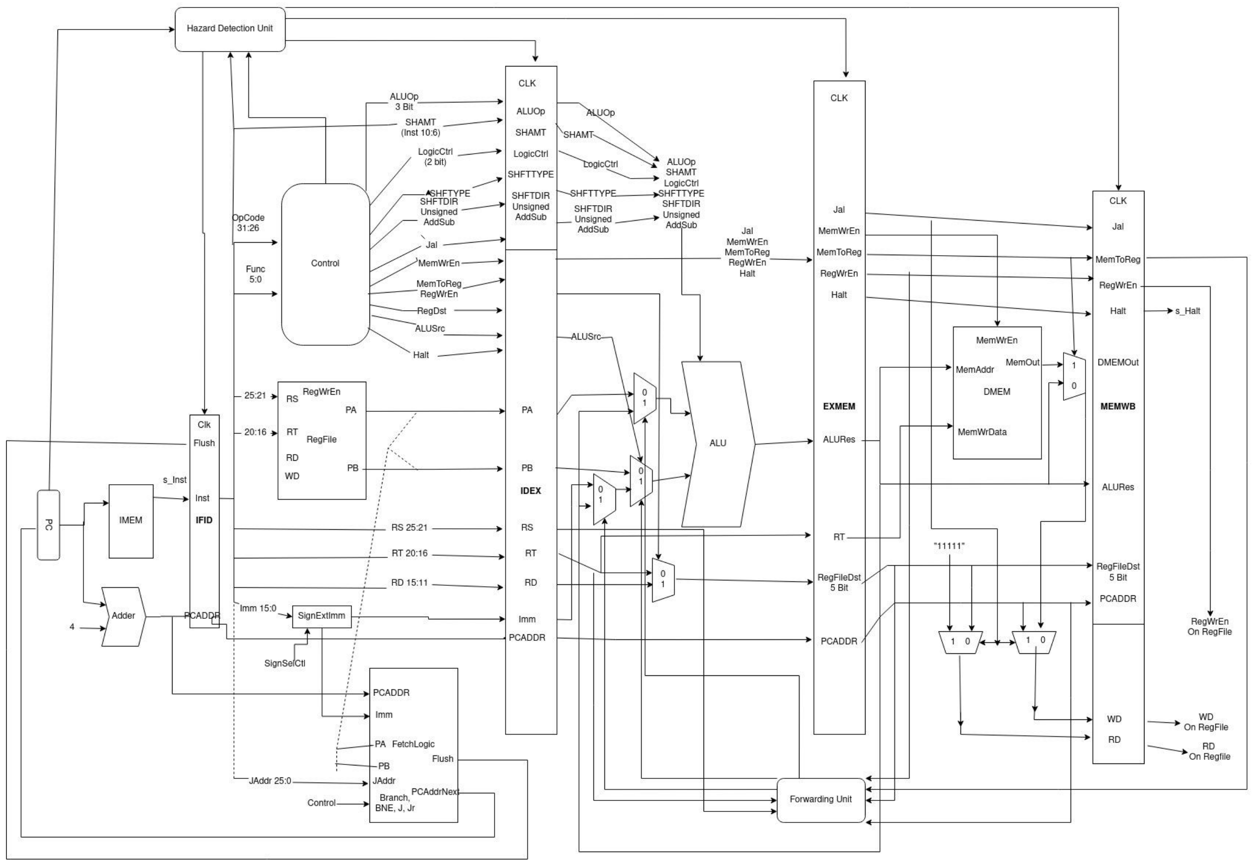
IF/ID: Load Use Stall, Jump/Jal flush, Jr stall, Branch stall

ID/EX: Load Use Flush, Jr flush, Branch flush

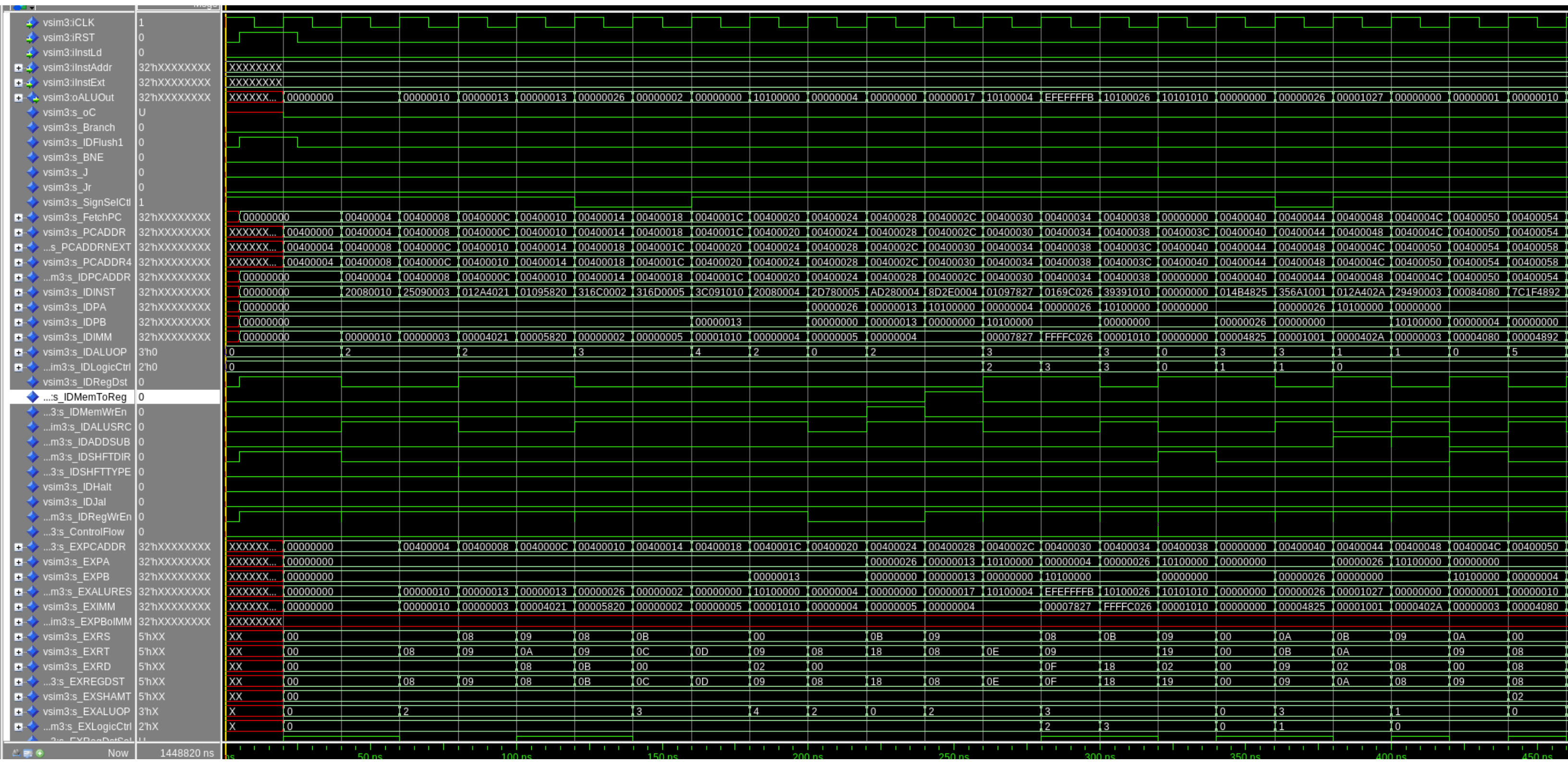
EX/MEM: ALU data gets forwarded from here

MEM/WB: Location for register can be forwarded from here

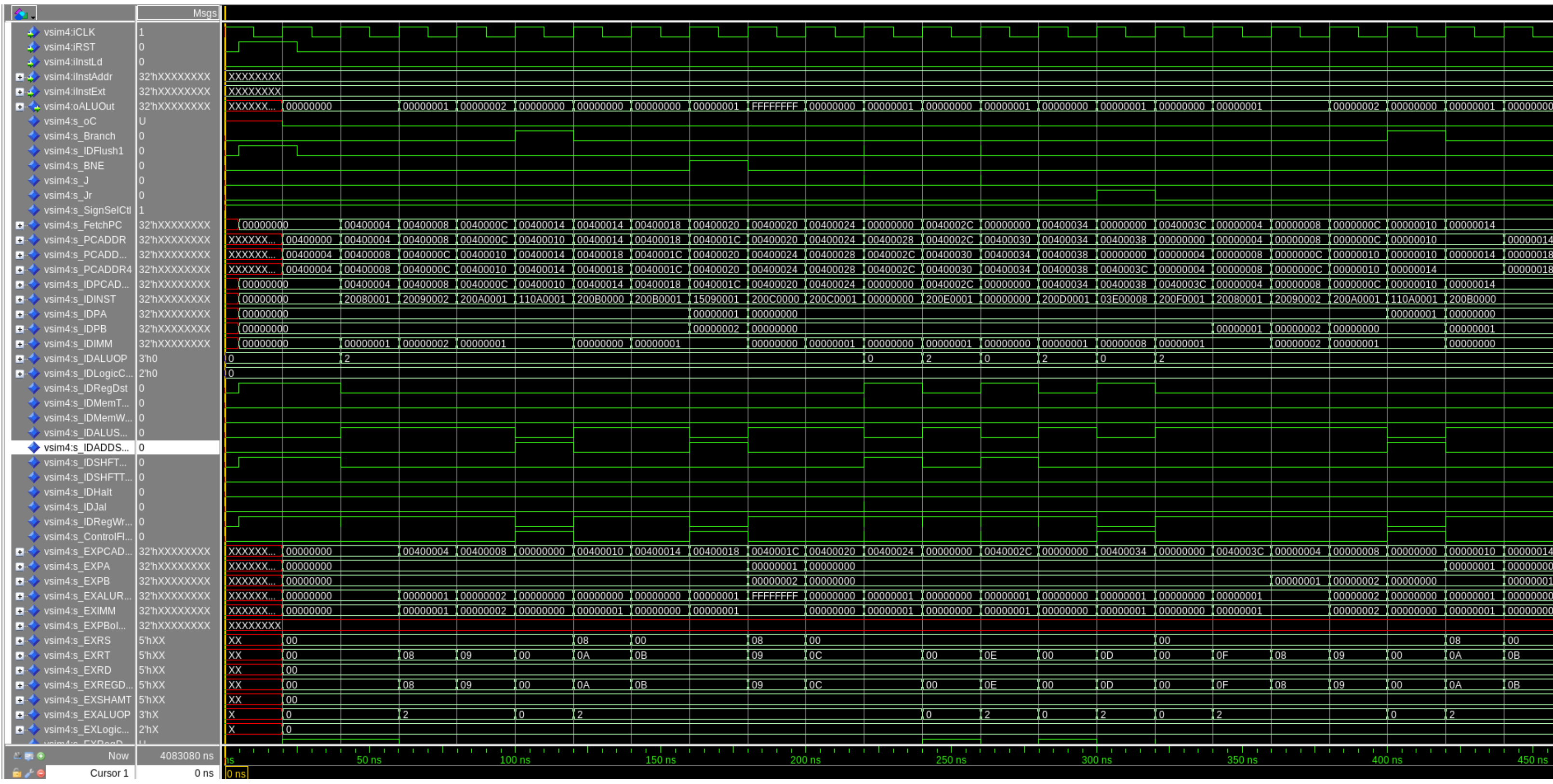
[2.d] implement the hardware-scheduled pipeline using only structural VHDL. As with the previous processors that you have implemented, start with a high-level schematic drawing of the interconnection between components.



[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.



Here is part of our waveform for test i, the forwarding test. The full waveform is saved as fowarding_vsim.wlf in our Proj2 Docs folder. We were not able to make it fully work. This is a result of a faulty forwarding unit. We tried to cover the cases, but we are missing something and out of time. The modelsim dump is in the same folder named modelsim_dump_2_i. For the second test, the results for the second test is the same. We were unable to make it fully work, the modelsim dump is in the same folder named modelsim_dump_2_ii and the waveform is named hazard_control_vsim.wlf. Here is a snippet of the waveform.



```
                     Msgs
vsim4:iCLK           1
vsim4:iRST           0
vsim4:iInstLd        0
vsim4:iInstAddr      32'hXXXXXXXX
vsim4:iInstExt       32'hXXXXXXXX
vsim4:oALUOut        32'hXXXXXXXX
vsim4:s_oC           U
vsim4:s_Branch       0
vsim4:s_IDFlush1     0
vsim4:s_BNE          0
vsim4:s_J            0
vsim4:s_Jr           0
vsim4:s_SignSelCtl   1
vsim4:s_FetchPC      32'hXXXXXXXX
vsim4:s_PCADDR       32'hXXXXXXXX
vsim4:s_PCADD...     32'hXXXXXXXX
vsim4:s_PCADDR4      32'hXXXXXXXX
vsim4:s_IDPCAD...    32'hXXXXXXXX
vsim4:s_IDINST       32'hXXXXXXXX
vsim4:s_IDPA         32'hXXXXXXXX
vsim4:s_IDPB         32'hXXXXXXXX
vsim4:s_IDIMM        32'hXXXXXXXX
vsim4:s_IDALUOP      3'h0
vsim4:s_IDLogicC...  2'h0
vsim4:s_IDRegDst     0
vsim4:s_IDMemT...    0
vsim4:s_IDMemW...    0
vsim4:s_IDALUS...    0
vsim4:s_IDADDS...    0
vsim4:s_IDSHFT...    0
vsim4:s_IDSHFTT...   0
vsim4:s_IDHalt       0
vsim4:s_IDJal        0
vsim4:s_IDRegWr...   0
vsim4:s_ControlFl... 0
vsim4:s_EXPCAD...    32'hXXXXXXXX
vsim4:s_EXPA         32'hXXXXXXXX
vsim4:s_EXPB         32'hXXXXXXXX
vsim4:s_EXALUR...    32'hXXXXXXXX
vsim4:s_EXIMM        32'hXXXXXXXX
vsim4:s_EXPBol...    32'hXXXXXXXX
vsim4:s_EXRS         5'hXX
vsim4:s_EXRT         5'hXX
vsim4:s_EXRD         5'hXX
vsim4:s_EXREGD...    5'hXX
vsim4:s_EXSHAMT      5'hXX
vsim4:s_EXALUOP      3'hX
vsim4:s_EXLogic...   2'hX
                Now  4083080 ns
           Cursor 1  0 ns
```

[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.

ID/EX.RegisterRd = IF/ID.RegisterRs

Inst / Cycle	1	2	3	4	5	6	
addi $t0, $0, 255	IF	ID	EX	MEM	WB		Rd = $t0
addi $t1, $t0, 255		IF	ID	EX	MEM	WB	Rs = $t0

ID/EX.RegisterRd = IF/ID.RegisterRt

Inst / Cycle	1	2	3	4	5	6	
addi $t2, $0, 255	IF	ID	EX	MEM	WB		Rd = $t2
add $t3, $0, $t2		IF	ID	EX	MEM	WB	Rt = $t2

EX/MEM.RegisterRd = IF/ID.RegisterRs

Inst / Cycle	1	2	3	4	5	6	7	
addi $t4, $0, 255	IF	ID	EX	MEM	WB			Rd = $t4
addi $t5, $0, 127		IF	ID	EX	MEM	WB		
addi $t5, $t4, 255			IF	ID	EX	MEM	WB	Rs = $t4

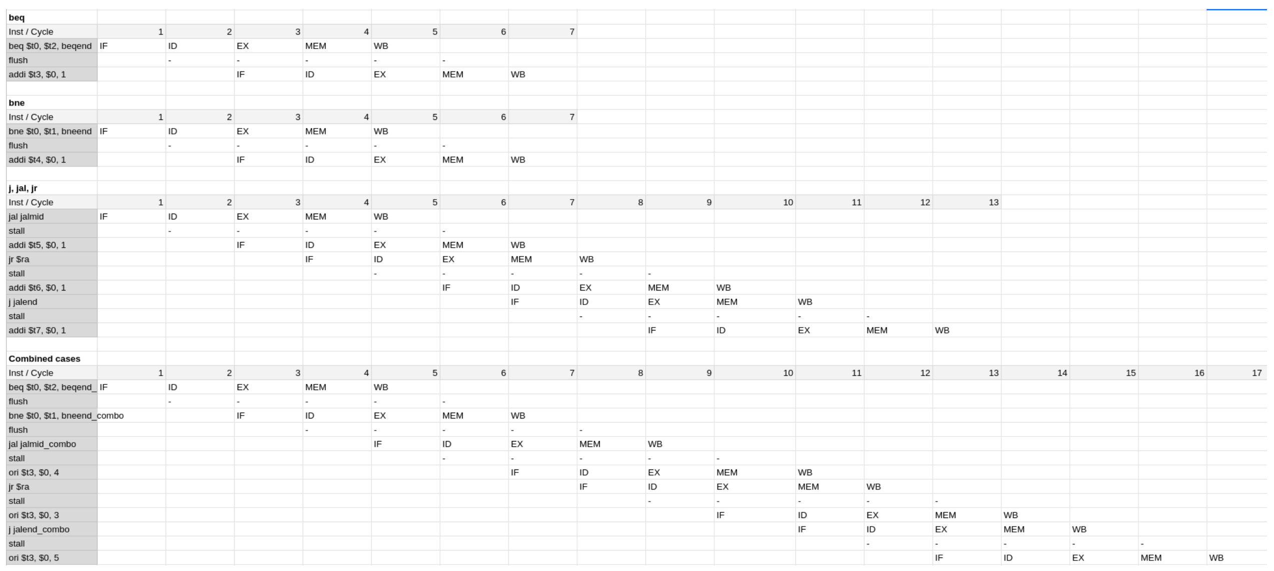
EX/MEM.RegisterRd = IF/ID.RegisterRt

Inst / Cycle	1	2	3	4	5	6	7	
addi $t6, $0, 255	IF	ID	EX	MEM	WB			Rd = $t6
addi $t7, $0, 127		IF	ID	EX	MEM	WB		
add $t7, $0, $t6			IF	ID	EX	MEM	WB	Rt = $t6

Combined cases

Inst / Cycle	1	2	3	4	5	6	7	8	9	
addi $t0, $0, 255	IF	ID	EX	MEM	WB					Rd = $t0
addi $t1, $t0, 255		IF	ID	EX	MEM	WB				Rs = $t0
addi $t1, $t0, 255			IF	ID	EX	MEM	WB			Rs = $t0, Rd = $t1
add $t2, $0, $t1				IF	ID	EX	MEM	WB		Rt = $t1
add $t2, $0, $t1					IF	ID	EX	MEM	WB	Rt = $t1

[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.



beq							
Inst / Cycle	1	2	3	4	5	6	7
beq $t0, $t2, beqend	IF	ID	EX	MEM	WB		
flush		-	-	-	-	-	
addi $t3, $0, 1			IF	ID	EX	MEM	WB

bne							
Inst / Cycle	1	2	3	4	5	6	7
bne $t0, $t1, bneend	IF	ID	EX	MEM	WB		
flush		-	-	-	-	-	
addi $t4, $0, 1			IF	ID	EX	MEM	WB

j, jal, jr													
Inst / Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
jal jalmid	IF	ID	EX	MEM	WB								
stall		-	-	-	-	-							
addi $t5, $0, 1			IF	ID	EX	MEM	WB						
jr $ra				IF	ID	EX	MEM	WB					
stall					-	-	-	-	-				
addi $t6, $0, 1						IF	ID	EX	MEM	WB			
j jalend							IF	ID	EX	MEM	WB		
stall								-	-	-	-	-	
addi $t7, $0, 1									IF	ID	EX	MEM	WB

Combined cases																	
Inst / Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
beq $t0, $t2, beqend_	IF	ID	EX	MEM	WB												
flush		-	-	-	-	-											
bne $t0, $t1, bneend_combo			IF	ID	EX	MEM	WB										
flush				-	-	-	-	-									
jal jalmid_combo					IF	ID	EX	MEM	WB								
stall						-	-	-	-	-							
ori $t3, $0, 4							IF	ID	EX	MEM	WB						
jr $ra								IF	ID	EX	MEM	WB					
stall									-	-	-	-	-				
ori $t3, $0, 3										IF	ID	EX	MEM	WB			
j jalend_combo											IF	ID	EX	MEM	WB		
stall												-	-	-	-	-	
ori $t3, $0, 5													IF	ID	EX	MEM	WB

[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: 48.92 mhz
Critical path: IMEM > MUXes Within the RegFile > ADDERSUB to get the Difference between the 2 ports > Setting the Zero with dataflow from the adder result > Or for the 2 branch conditionals > MUXPCADD to see if taking branch conditional or PC+4 > Write Back to the PC.