## CprE 381: Computer Organization and Assembly-Level Programming

## **Project Part 2 Report**

Team Members: Josh Arceo

Joseph Barnes III

Evan Shiber

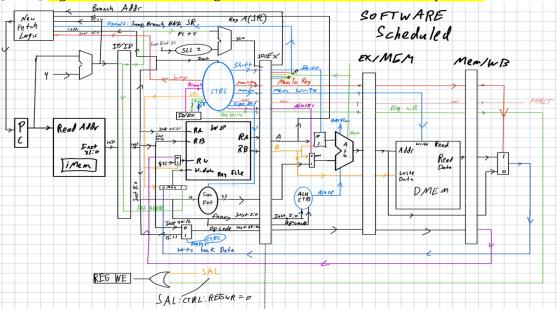
Project Team Group #: Sec C 03

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

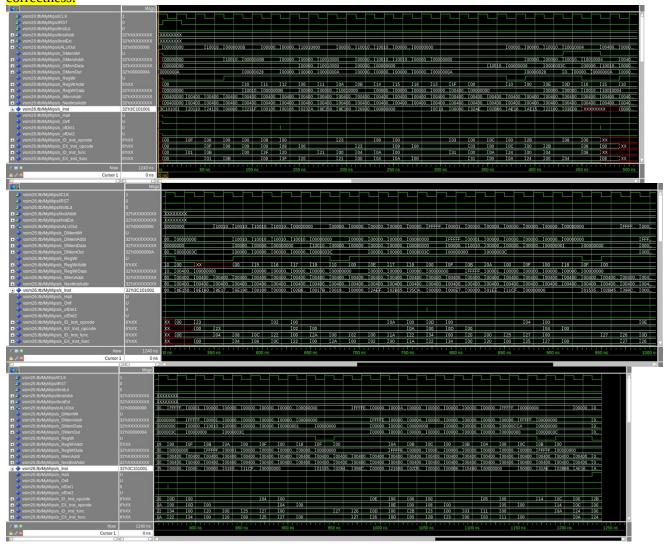
[1.a] Come up with a global list of the datapath values and control signals that are required during each pipeline stage.

auring cac	ii pipeiiie s	uge.					
DECODE		EXECUTE		MEMORY		WRITEBACK	
HALT	GENERATED	HALT		HALT		HALT	CONSUMED
		OVERFLOW	GENERATED	OVERFLOW		OVERFLOW	CONSUMED
MEM TO REG	GENERATED	MEM TO REG		MEM TO REG		MEM TO REG	CONSUMED
REG WRITE	GENERATED	REG WRITE		REG WRITE		REG WRITE	CONSUMED
MEM WRITE	GENERATED	MEM WRITE		MEM WRITE	CONSUMED	)	
ALU SRC	GENERATED	ALU SRC	CONSUMED				
SHIFT	GENERATED	SHIFT	CONSUMED				
BRANCH	GENERATED	BRANCH	CONSUMED				
BRANCH NE	GENERATED	BRANCH NE	CONSUMED				
JUMP RET	GENERATED	JUMP RET	CONSUMED				
JUMP GEN	+CONSUMED						
JAL GEI	N+CONSUMED						
	N+CONSUMED						

[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.



We can see the current Instrution in the IF stage in s\_Inst, and since it's software scheduled they're is not flushing or stalling so we can assume that in the next cycle it moves to ID, in the 1'st image the first instruction is lui \$s0, 0x1001, then 4 cycles later in it's WB (100ns) stage we see RegWrAddr is 10 (\$s0) and RegWrdata is 0x10010000. We can follow the rest of the waveform to see all writes will be properly processed.

JAL (300ns) has 3 NOPs ahead of it to ensure all r type instructions before it have finished using the regfile to write and then JAL can jump and write the correct addr into \$31 in the ID stage.

Branches have 2 nops after them to correctly calculate whether it will branch or not before following next instrucitons. (850 ns)

[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.

⇒ vsim27:/tb/MyMips/iCLK	1																						
vsim27:/tb/MyMips/iRST	0	حصر																					
vsim27:/tb/MyMips/iInstLd	0									$\blacksquare$	_							_	$\rightarrow$	$\rightarrow$	_	$\overline{}$	
vsim27:/tb/MyMips/iInstAddr	32hXXXXXXXXXX	XXXXXXXX																					
vsim27:/tb/MyMips/iInstExt	32hXXXXXXXXX	XXXXXXXXX														-			V				
vsim27:/tb/MyMips/oALUOut	32h00000000	b I 00000000	) 100	0000 IFF	FFF I 000	00000	(100)	10 🕻 100	101000	00000	(10010.	1000001	00000100	0000100	00000	(0000)	0000		(10010	0 (1001	0)(100)	o X0000	0 [Ff
vsim27:/tb/MyMips/s_DMemWr	U				_				_					_					_				
vsim27:/tb/MyMips/s_DMemAddr	32hXXXXXXXXX	p 1000001	00000000	100		FF 1000				1010000	0000	(10010)	0000010	0000000	_		=	=	=		0)1001		
vsim27:/tb/MyMips/s_DMemData	32hXXXXXXXXXX	00000000		_		001000				1010000		(00000)			0001000	00000	=	=	$\rightarrow$		00)		
vsim27:/tb/MyMips/s_DMemOut	32'h0000000A	D000000A		_	1000	00 I 000	0000A	1000	000100	00I0000	000A	(00000)	0000000A					_		101	00000[	0 10000	0100
vsim27:/tb/MyMips/s_RegWr	U	Ļ			Ins	<u> </u>	100													$\overline{}$			
vsim27:/tb/MyMips/s_RegWrAddr	5ħXX	00							[13		00		14 0			(00	=		$\rightarrow$	$\rightarrow$	13	(14	. Ya
vsim27:/tbi/MyMips/s_RegWrData     vsim27:/tbi/MyMips/s_IMemAddr	32hXXXXXXXX 32hXXXXXXXX	D [00400]	100000 100		1000	)00 [ FFF	+ 1000¢	00000	1000	0011001	) 100000C	100	0000010	0000 100	000000	. 7.0.0	Y	. V.	. V.	. Y		0 (1001	
vsim27:/tb/MyMips/s_IMemAddr		D 100400	00400100	0400100	4001004	00 1004	00 1 0040	00 ( 004	00   004	0010040	) 100400.	100400 1	00400100	1400100	001004	00-10040	)10040	0 1 0040	00 <u>(</u> 0040)	0 ( 0040	00) 0040	0 10040	0 100
vsim27:/tb/MyMips/s_NextInstAddr	32hXXXXXXXXX	0 [00400]																					
vsim27:/tb/MyMips/s_inst	32h3C101001	0 [21080]	.1532F100	0000000	18D	73 I 216	3010000	00000	18D7	4 12129	100000.	X02746 X	00000000	1,15	001000	200000	LAD75	3 LAD7	4 (8D74	12108	3011532	F 10000	0000
vsim27:/tb/MyMips/s_Halt	U																						
vsim27:/tb/MyMips/s_Ovfl	U																	_					
vsim27:/tb/MyMips/s_ofDel1	0																						
vsim27:/tb/MyMips/s_ofDel2	U																						
vsim27:/tb/MyMips/s_ID_Inst_opcode	6hXX		108 105			[23				[23	(08	(00			(05	(00)		(28		(23		05	(00
vsim27:/tb/MyMips/s_EX_Inst_opcode	6hXX	05 [00	1,08				(23	(08	[00		23		00			(05	(00		(28		23	80)	(05
vsim27:/tb/MyMips/s_ID_Inst_func	6ħXX		IO1   I2E				(04				(01	(00			105	(00			(3C	(00	(01	X2E	700
vsim27:/tb/MyMips/s_EX_Inst_func	6hXX	05 IO0	101	1 I2E	100			(04	100			(01	00 12	100		(05	(00			(3C	00)	(01	(2)
	Now 5720 ns	500 ns			1111	1111			650 ns			1111				1111	1111		1111	111	1111	1111	
	rsor 1 0 ns	500 ns		550 ns					650 ns		700 ns		750 n		800			850 ns		900			950 n:
Cu Cu	SOF 1 O ns		_																				
	Msgs			_																			
	Msgs						=	=	_						_			_		_	=	_	
ysim27:/tb/MyMips/iCLK	1																						
ysim27:/tb/MyMips/iRST	0																						
♦ vsim27:/tb/MyMips/iInstLd	0														الصد								
vsim27:/tb/MyMips/iInstAddr	32hXXXXXXXXX	XXXXXXX																					
vsim27:/tb/MyMips/iInstExt	32'hXXXXXXXXXXX	XXXXXXXX													السنا								
vsim27:/tb/MyMips/oALUOut	32'h00000000	0 X 10010 X	10010100	000000	(100	10 (0000	0100001	010000	0 10000	0000	100000	100000000	(00	000(FFE	F) 0000	0000	(10010	[100]	0100000	000	(10010	(00000	(000
vsim27:/tb/MyMips/s_DMemWr	U	_																					
vsim27:/tb/MyMips/s_DMemAddr	32hXXXXXXXXXX	00000000 Y	10010 [100	0101000	00000	11001	0100001	010000	010000	o I 00000	000	10000010	0000000	(000	00) FFFF	F 100000	000	11001	0110010	100000	0000	[10010	1000
vsim27:/tb/MyMips/s DMemData	32hXXXXXXXXX	00000000 X	000001100	0101000	00000	10000	0100001	010000	010000	o100000	000			1000	00) 0000	100000	000	10000	0110010	100000	0000	(00000	1000
vsim27:/tb/MyMips/s_DMemOut	321100000000A	0000000A (	00000100	0001000	0000A	(0000	0[0000]	000A							0000	0)00000	00A	10000	0100000	100000	A000	(00000	)(00
→ vsim27:/tb/MyMips/s_RegWr	U																						
vsim27:/tb/MyMips/s_RegWrAddr	5ħXX	12 100	13	IQB	) 00		[14	109	100	Ioc	100				708	112	100		113	IOB	100		114
vsim27:/tb/MyMips/s_RegWrData	32hXXXXXXXXX	F 100000000	100	000	10 1000	00000				o I 00000	Locopooi	n Yr	0000(00	000000	) 0000	))FFFFF	Young	0000	Loonno	110010	100000	000	100
vsim27:/tb/MyMips/s_IMemAddr	32'hXXXXXXXXX	0 X00400 X																					
vsim27:/tb/MyMios/s NextInstAddr	32hXXXXXXXXX	0 100400 Ye																					
→ vsim27:/tb/MyMips/s_Inst	32%3C101001	2100000000		74 1212						0100000		[21080]				21680					100000		
▼ vsim27:/tb/MyMips/s Halt	U	-E XVVVVVVV	100	//	30	VV AUE / 14	010000	****	1,1000	V	VVV	121000	33EF 190	22000	7,0075	- AZIOĐO		AVV	100/14	121270	2	102740	
▼ vsim27:/tb/MvMips/s_Hait     ▼ vsim27:/tb/MvMips/s Ovfl	U																						
√ vsim27:tb/MyMips/s_ofDel1	0																						
	U																						
vsim27:/tb/MyMips/s_ofDel2	6hXX	23 I 08 I		-		<b>—</b>			=	105						123	7			7			
vsim27:/tb/MyMips/s_ID_Inst_opcode					)(08 )(23	(00 Yos	100		=	105	(00		8 (05		Van	123	08	[00]	Yes	23	123	(00)	Y00
vsim27:/tb/MyMips/s_EX_Inst_opcode	6hXX			_					=	7		100	X08			$\rightarrow$	(23		100	$\rightarrow$			
vsim27:/tb/MyMips/s_ID_Inst_func	6hXX	00 X04 X			(01		I2A			105	[00]	- 10	1 X2E	100	_,	_	(04	100		$\rightarrow$	[01	(00)	
vsim27:/tb/MyMips/s_EX_Inst_func	6hXX	00 X	04 100			(01	100	I2A	100		(05	100	(01	(2E	),00			[04	100			(01	(00
	Now 5720 ns		1050 ns		110	1111	1111	1150 ns	1 1 1 1	1200 r	1111	111111		130	1111	1111	050	1111	1 1 1000		1111	450 ns	
0	rsor 1 0 ns			5				1150 NS						130					1400 F				
Co	[38]																						
	Msgs		_																				
A		_					<b>—</b>	7	- E	- E						_		-					
⇒ vsim27:/tb/MyMips/iCLK ⇒ vsim27:/tb/MyMips/iRST	0																					-	_
vsim27:/tb/MyMips/iRST vsim27:/tb/MyMips/instLd																							
	0	MANAGEMENT																					
vsim27:/tb/MyMips/iInstAddr	32hXXXXXXXX 32hXXXXXXXXX	XXXXXXXXX																					
vsim27:/tb/MyMips/iinstExt								V						V	V			V	V		Y		
vsim27:/tb/MyMips/oALUOut	32'h00000000	100000100	00010000	00000	100000	000		(10010	110010	110010	(00000 I	FFFF1000	00000	110010	(10010	10000000		10010	(00000)	00000	100000	10000000	0
vsim27:/tb/MyMips/s_DMemWr	U																						
vsim27:/tb/MyMips/s_DMemAddr	32hXXXXXXXXX	100000100										00000IFFF				110010			(10010				
vsim27:/tb/MyMips/s_DMemData	32hXXXXXXXX	100000100		00000100	000							000b01000											10000
vsim27:/tb/MyMips/s_DMemOut	32110000000A	100000100	00000A						10100	00010	0000010	000001000	00 I 0000	A000	(00000	100000	(0000000	)A	(00000)	00000	10000000	A	
vsim27:/tb/MyMips/s_RegWr	U																						
	5ħXX	114 109		[00]	[00]						14	108	(12	(00)				(00		14	(09	100	L0C
→ voinie i naimymips/s_regwindai	32'hXXXXXXXXXXX	100000100										00000				100000					00000		
vsim27:/tb/MyMips/s_RegWrData	32hXXXXXXXXX	1004001004	400 10040	0[00400		(00400	X00400	(00400	100400.	100400	(00400I	004001004	0010040	0 (00400	(00400	I00400	00400	00400	(00400)	00400	100400	I00400	10040
vsim27:/tb/MyMips/s_RegWrData vsim27:/tb/MyMips/s_IMemAddr		1004001004	400 10040	0[00400		100400	(00400	X00400	100400	100400	00400Id	004001004	0010040	0 (00400	(00400	100400	00400	(00400	(00400	00400	100400	100400	10040
vsim27:tb/MyMips/s_RegWrData vsim27:tb/MyMips/s_IMemAddr vsim27:tb/MyMips/s_NextInstAddr	32hXXXXXXXXX			000000						1532F			3 1216B			[8D74						15800	
vsim27:tb/MyMips/s_RegWrData vsim27:tb/MyMips/s_IMemAddr vsim27:tb/MyMips/s_NextInstAddr	32hXXXXXXXX 32h3C101001	1000000000				الراجعي الأحد			تزي														
vsim27:tbrMyMips/s_RegWrData vsim27:tbrMyMips/s_IMemAddr vsim27:tbrMyMips/s_NextinstAddr vsim27:tbrMyMips/s_Inst		100000000																					
vsim27:itb/MyMips/s_RegWirData vsim27:itb/MyMips/s_MemAddr vsim27:itb/MyMips/s_NextinstAddr vsim27:itb/MyMips/s_linst vsim27:itb/MyMips/s_Halt	32h3C101001	100000000																					
vsim27zbbNyMlps/s, RegWhData     vsim27zbbNyMlps/s, RegWbData     vsim27zbbNyMlps/s, IMemAddr     vsim27zbbNyMlps/s, INextinstAddr     vsim27zbbNyMlps/s, Inst     vsim27zbbNyMlps/s, Inst     vsim27zbbNyMlps/s, Ovl     vsim27zbbNyMlps/s, Ovl     vsim27zbbNyMlps/s, Ovl     vsim27zbbNyMlps/s, Ovl     vsim27zbbNyMlps/s, Ovl     vsim27zbbNyMlps/s, Ovl	32h3C101001 U	100000000																					
vsim27zhbMyMipsis_RegWData vsim27zhbMyMipsis_MemAddr vsim27zhbMyMipsis_NextinstAddr vsim27zhbMyMipsis_inst vsim27zhbMyMipsis_inst vsim27zhbMyMipsis_Ovfil vsim27zhbMyMipsis_Ovfil vsim27zhbMyMipsis_Otfil	32h3C101001 U U	100000000																					
wsim27zhbh/Mipsis_ReglWData vsim27zhbh/Mipsis_NemAddr vsim27zhbh/Mipsis_NextinsAddr vsim27zhbh/Mipsis_NextinsAddr vsim27zhbh/Mipsis_Inst vsim27zhbh/Mipsis_Ovil vsim27zhbh/Mipsis_Ovil vsim27zhbh/Mipsis_Ovil vsim27zhbh/Mipsis_Othel1 vsim27zhbh/Mipsis_Othel1 vsim27zhbh/Mipsis_Othel2	32h3C101001 U U 0	100000000		105	100		Y28		123	Yos	05 10	10	¥23	Yos	Yoo		(23	Yos	Yoo				105
vsim272hbMydipsis_RegWhData vsim272hbMydipsis_MemAddr vsim272hbMydipsis_MemAddr vsim272hbMydipsis_NewtinstAddr vsim272hbMydipsis_NextinstAddr vsim272hbMydipsis_Next vsim272hbMydipsis_Next vsim272hbMydipsis_OPIII vsim272hbMydipsis_OPIIII vsim272hbMydipsis_OPIIII vsim272hbMydipsis_OPIIII vsim272hbMydipsis_OPIIII vsim27hbMydipsis_OPIIII vsim27hbMydipsis_OPIIIII vsim27hbMydipsis_OPIIII vsim27hbMydipsis_OPIIIII vsim27hbMydipsis_OPIIII vsim	32%3C101001 U U 0 U 6hXX	00		(05			<u>(</u> 28													100			[05
wisin27.20bMydpsgs_RegWhData           wisin27.20bMydpsgs_MemAddf           wisin27.20bMydpsgs_MemAddf           wisin27.20bMydpsgs_MemAddf           wisin27.20bMydpsgs_Modes_MemAddf           wisin27.20bMydpsgs_ODel1           wisin27.20bMydpsgs_ODel1           wisin27.20bMydpsgs_ODel0           wisin27.20bMydpsgs_ODel0           wisin27.20bMydpsgs_ODel0           wisin27.20bMydpsgs_DDel0           wisin27.20bMydpsgs_DDel0	32h3C101001 U U 0 U 6hXX 6hXX	00		- 1	105	χοο		(2B		)23	08	5 100		(23	(08	100		) 123	Y08	100	Y00		
wimir2.7hbh/Mpuis, Reqwidotas wimir2.7hbh/Mpuis, Memdidd wimir2.7hbh/Mpuis, Memdidd wimir2.7hbh/Mpuis, Memdidd wimir2.7hbh/Mpuis, Memdidd wimir2.7hbh/Mpuis, Met wimir2.7hbh/Mpuis, Met wimir2.7hbh/Mpuis, Gibel wimir2.7hb/Mpuis, Gibel wimir2.7hbh/Mpuis, Gibel wimir2.7hbh/	32h3C101001 U U 0 U 6hXX 6hXX	00 100 12A 100		[05 [05	I05 I00			(2B (3C		)23	(08 IC	05 I00	,		(08	100		) 123	Y08		100		***
wimin27.hbb/Mpuis, Regwidota wimin27.hbb/Mpuis, Memfaddr wimin27.hbb/Mpuis, Memfaddr wimin27.hbb/Mpuis, Memfaddr wimin27.hbb/Mpuis, Neimin4Addr wimin27.hbb/Mpuis, Neimin4Addr wimin27.hbb/Mpuis, Neimin27.hbb/Mpuis, Meil wimin27.hbb/Mpuis, Gilleil wimin27.hbb/Mpuis, Gilleil wimin27.hbb/Mpuis, Gilleil wimin27.hbb/Mpuis, Gilleil wimin27.hbb/Mpuis, Dilleil wimin27.hbb/Mpuis	32h3C101001 U U 0 U 6hXX 6hXX	00		- 1	105			(2B (3C		)23 )01	08	05 I00	,	(23	(08			) 123			100 12A		[05 [05
vsim27zhbMyMipsis_RegWData vsim27zhbMyMipsis_MemAddr vsim27zhbMyMipsis_NextinstAddr vsim27zhbMyMipsis_inst vsim27zhbMyMipsis_inst vsim27zhbMyMipsis_Ovfil vsim27zhbMyMipsis_Ovfil vsim27zhbMyMipsis_Otfil	32h3C101001 U U 0 U 6hXX 6hXX	00 100 12A 100 100 12A		[05	I05 I00			(28 (3C	I00 I3C	)23	(08   10 (2E   10 (01   12	05 I00		(23	(08	100		X23 X01	Y08		100 12A		***

The array is properly sorted at the end of execution. Which can be seen when we compare Dmem values at the beginning vs end of execution. In our cond label instead of adding NOPs after incrementing \$t0 before using it in a branch comparison, we instead moved it to the beginning of the label and moved another branch comparison after it to give room before we compared it. (1200ns) Similarly, we moved addi \$t0 to the beginning of the print\_loop only requiring 1 nop instead of 2 before the branch condition. This did require us to add a nop before this loop however, since it isn't inside of the loop it actually saves us 1 cycle every time that loop runs. (This part is near the end and isn't shown in the waveform)

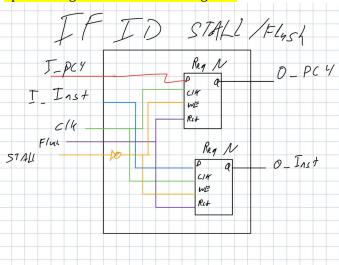
[1.d] report the maximum frequency your software-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).

50.39mhz

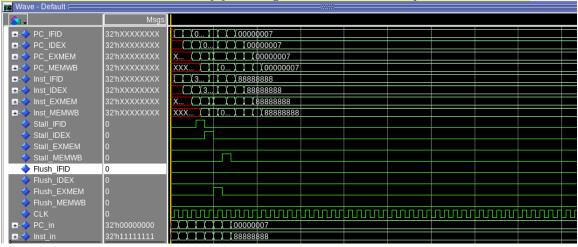
Critical Path:

PC\_Reg  $\rightarrow$  iMem  $\rightarrow$  IFID  $\rightarrow$  PC Fetch  $\rightarrow$  Reg File  $\rightarrow$  IDEX  $\rightarrow$  ALU A MUX/ALU B MUX  $\rightarrow$  ALU  $\rightarrow$  EXMEM  $\rightarrow$  DMEM  $\rightarrow$  MEMWB  $\rightarrow$  WB MUX  $\rightarrow$  RegDst Mux  $\rightarrow$  JAL MUX

[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.

add/addi/addiu ALU Out → DMEM ADDR → WB

and/andi/nor/xor/xori/or/ori ALU Out  $\rightarrow$  DMEM ADDR  $\rightarrow$  WB lui/lw ALU Out  $\rightarrow$  DMEM ADDR // DMEM Out  $\rightarrow$  WB

slt/slti ALU Out  $\rightarrow$  DMEM ADDR  $\rightarrow$  WB sll/srl/sra ALU Out  $\rightarrow$  DMEM ADDR  $\rightarrow$  WB sub/subu ALU Out  $\rightarrow$  DMEM ADDR  $\rightarrow$  WB

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

add Consumes RS, RT → RegA/RegB → ALU A/ALU B

addi/addiu Consumes RS → RegA → ALU A

and/nor/xor/or/ Consumes RS, RT → RegA/RegB → ALU A/ALU B

andi/ xori/ ori Consumes RS → RegA → ALU A lui Consumes imm → ALU A (shifted)

lw Consumes RS  $\rightarrow$  ALU A  $\rightarrow$  ALU OUT  $\rightarrow$  DMEM ADDR sw Consumes RS  $\rightarrow$  ALU A  $\rightarrow$  ALU OUT  $\rightarrow$  DMEM ADDR Then

Consumes RD → DMEM DATA

slt Consumes RS, RT → RegA/RegB → ALU A/ALU B

slti Consumes RS → RegA → ALU A

sll/srl/sra Consumes RS, RT  $\rightarrow$  RegA/RegB  $\rightarrow$  ALU A/ALU B sub/subu Consumes RS, RT  $\rightarrow$  RegA/RegB  $\rightarrow$  ALU A/ALU B jr Consumes RS  $\rightarrow$  RegA Out  $\rightarrow$  Next Addr

[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.

Instruction that writes to a register followed by an instruction that consumes the value of that register as either the next instruction or the 2<sup>nd</sup> instruction after.

Instruction lw that writes to a register followed by an instruction that consumes that register value, for this we will need one stall to ensure that lw can read the data from memory then it can be forwarded from MEM to the EX.

Instruction that writes to a register followed by LW that uses that register as the dataAddr, the value can be forwarded like the first data dependency listed without stalling

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

FETCH	DECODE		EXECUTE		MEMORY		WRITEBACK	
	HALT	GENERATED	HALT		HALT		HALT	CONSUMED
PC Stall			OVERFLOW	GENERATED	OVERFLOW		OVERFLOW	CONSUMED
	MEM TO REC	GENERATED	MEM TO REG		MEM TO REG		MEM TO REG	CONSUMED
FID Stall	REG WRITE	GENERATED	REG WRITE		REG WRITE		REG WRITE	CONSUMED
FID Flush	MEM WRITE	GENERATED	MEM WRITE		MEM WRITE	CONSUMED		
	ALU SRC	GENERATED	ALU SRC	CONSUMED				
	SHIFT	GENERATED	SHIFT	CONSUMED	FWD Data	CONSUMED		
	BRANCH	GENERATED	BRANCH	CONSUMED				
	BRANCH NE	GENERATED	BRANCH NE	CONSUMED				
	JUMP RET	GENERATED	JUMP RET	CONSUMED				
	JUMP GEN	I+CONSUMED						
	JAL GE	N+CONSUMED	FWD A	GEN+CONSUMED				
	SIGNEXT GE	N+CONSUMED	FWD B	GEN+CONSUMED				
	IDEX Stall		EXMEM Stall		MEMWB Stall			
	IDEX Flush		EXMEM Flush		MEMWB Flush			

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

Jump: ID JAL: ID JR: EX BEQ: EX BNE: EX

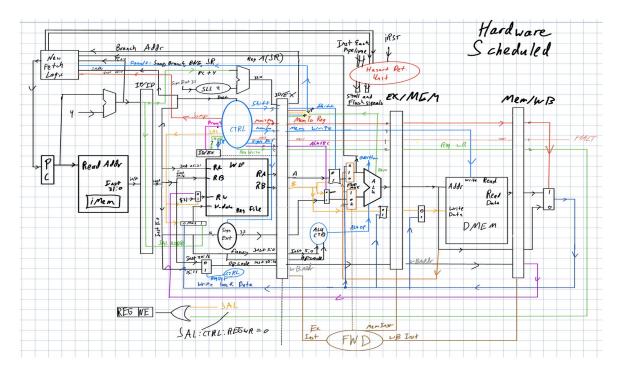
[2.c.ii] For these instructions, list which stages need to be stalled and which stages need to be squashed/flushed relative to the stage each of these instructions is in.

Jump: Stall PC and flush IFID when reaches EX

JAL: Stall in IF when another instruction is writing to reg, until they're done. Then stall IFID when in ID and flush IFID, IDEX, EXMEM when reaches EX. (Need to be done the way our JAL is implemented because it writes in ID and not WB)

JR: EX BEQ: EX BNE: EX

[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.

/ 1		_	
Test 1	Data Hazard	Add	Test read after write hazard with just adds/addis
Test 2	Data Hazard	Add/LW	Test read after write hazard with add/addi and lw
Test 3	Data Hazard	Add/LW/SW	Test read after write hazard with add/addi and lw and sw

## [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.

	ControlTest1	Control Hazard	BEQ	Tests to make sure BEQ branches correctly and does not run instructions that it branched past
	ControlTest2	Control Hazard	BEQ/JUMP	Tests to make sure it BEQ wont run jumps it branches past since jumps execute in ID
	ControlTest3	Used all Branch	Jump Base tests	These tests ensured comprenensive testing for all control hazards that we could come across
				and lead to us discovering a rare edge case in JR_5 where the program would flush the halt
				after a JR thinking that it was misread because the Jaddr was = PC + 4
Ī				

[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).

35.60mhz

PC\_Reg → iMem → IFID → Control → Hazard Detection → Reg File → IDEX → Forward → PC Fetch → FWD MUX A/FWD MUX B → ALU A MUX/ALU B MUX → ALU → EXMEM → FWD MUX DMEM Data → DMEM → MEMWB → WB MUX → RegDst Mux → JAL MUX