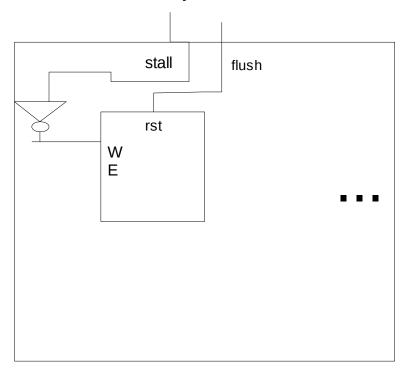
#### 0. Prelab

This is in the file omniscient spreadsheet.

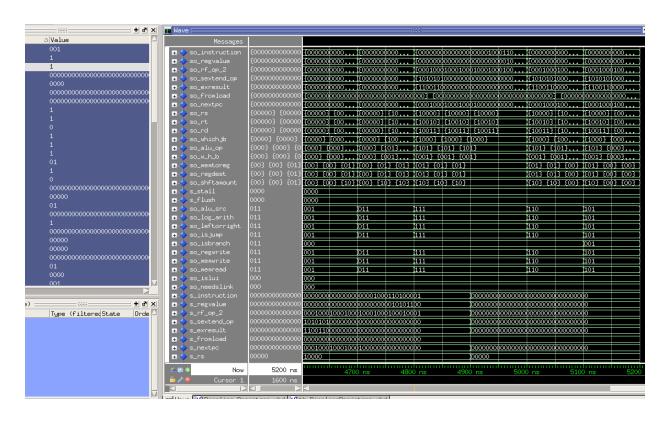
### 1. Pipelined Registers

a) To implement an ideal N bit register with stalling and flushing we can simply allow the stall and flush to access the reset and write enable bits. Then we just not the stall and set reset directly to the flush input and it would be implemented if it was ideal.



b) We created the file pipeline registers and a test bench which was verified by our TA. Results of the test bench:

New   Section	ave ====				*****				= 6
Section   Sect			1000100 00010001000	. 10000000000000000	000000000000000000000000000000000000000	0		100010001000	=
Comparison   Com	♦ s_rs	10000	10000	100000				10000	
Section   Sect	❤ s_rt ❖ s_rd								
September   Sept								1000	
### STANDARD   100		001						101	
# 1   10   10   10   10   10   10   10	s_memto								
		mount 10						10	
Second   S		rc 1							
Second   S	🤷 s_lefto	rright 1							
Second   S								-	
### ##################################	🤷 s_regwr	ite 1							
	◆ s_memre							<del>                                     </del>	
		link 0							
Secretary   Secr		1							
				, 1000000000000000000000000000000000000	00000100011010000	1		1000000000000	
	sf_rf_o	P_2 0000000	00000000	1000100010001000	1000100010001000	1		1000000000000	
So_Promised			000000000000000000000000000000000000000	[1010101@000000C	000000000000000000000000000000000000000	0		1000000000000	
se_Presided (000000000000 (000000000000000000000	◆ sf_from	load 0000000	0000000 0000000000000000000000000000000	000000000000000000000000000000000000000	000				
so_Fremional				. 1000100010001000	100000000000000000	0	V0C)C)C)C) 4 A 4 4 C		
so_nextpc so_nextpc so_re	<u>Y</u>								
so_ns   500003 [00000] [000003 [00000] [00000] [00000]   5000000]   500000]   500000]   500000]   50000000000	X = -								
so_rtd					, <u> </u>	000000000000000000000000000000000000000	<del>,,,,,,,,,</del> ,,,,,,,,,,,,,,,,,,,,,,,,,,,	,1000100010001	0000
so_nd	<u> </u>								
So_which,   Description   Source   So	· · · · · · · · · · · · · · · · · · ·								
So_alu_op   5000   5000   5000   5001   50						\$00013 \$000	1500003 5000	03 {1000}	
So_wh_b									
So_nemboreg   600   601   60									
So_regdest   \$003	so_memtoreg								
Substable   Oliforn   Ol	so_regdest	{00} {00} {01}	{01} {01} {01}						
Substable   Oliforn   Ol	so_shftamount	{00} {00} {10}	{01} {01} {10}			<b>(</b> 01} {00} {10}	<b>[{00} {00}</b> {	10}	
So_alu_src   So_log_arith   So_log	s_stall	0011	1010			0000			
So_log_arith   So_leftorright   So_lef	s_flush	0100	0000			0010			
So_leftorright   So_is_jump	so_alu_src	001	111			101	001		
So_is_jump   So_is_branch   So_is_	> so_log_arith	001	111			101	001		
So_isbranch   So_regwrite   O01   111   1101   1011   1001   1111   1111   1011   1001   1001   1111   1001   1001   1111   1001   10	> so_leftorright		111			101	001		
so_regwrite         001         111         101         001         001         111         101         001         001         001         111         1001         0001         0001         001         111         0001<	>so_isjump	001	111			101	001		
so_memwrite       001       111       101       001         so_memmread       001       111       101       001         so_needslink       000       000       000       000         s_instruction       00000000000000       000000000000000000000000000000000000	> so_isbranch								
so_memread       001       i11       i01       001       001       001       000	> so_regwrite								
so_islui       000									
so_needslink       000						101	001		
s_instruction s_regvalue s_rf_op_2 o0010001000100 001000100010001001101000000									
S_regvalue	_			24222442422					
S_rf_op_2									
s_sextend_op s_exresult 1001010000000 10101010000000000000000	-								
S_exresult   11001100000000									
s_framload									
S_nextpc									
s_rs   10000   10000   10010   10010   10010   10011   10011   10011   1000   1000   1011   1001   1011   1									
S_rt				333000000000					
s_rd									
s_which jb 1000 101 1000 101 101 101 101 101 101	< = .								
S_alu_op 101 101 101 101 101 101 101 101 101 10									
Now 3300 ns 00 ns 2500 ns 2600 ns 2700 ns 2800 ns 2900 ns 300 ns									
South in the state of the stat									
t⊜ Cursor 1 1600 ns	Now Now	3300 ns	00 ns 2500	ns 2600	ns 270	9 ns 280	00 ns	2900 ns	3
	😊 Cursor 1	1600 ns							



## 2. Data Dependencies

All parts number two can be seen in the Omniscient Spreadsheet on the last three tabs.

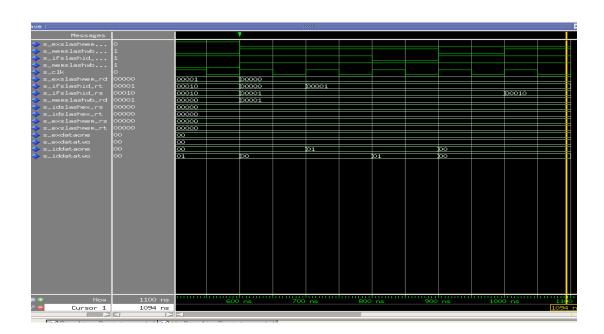
## 3. Forwarding and Hazard Detection

Shown in the Dependencies tab of the Omniscient Spreadsheet.

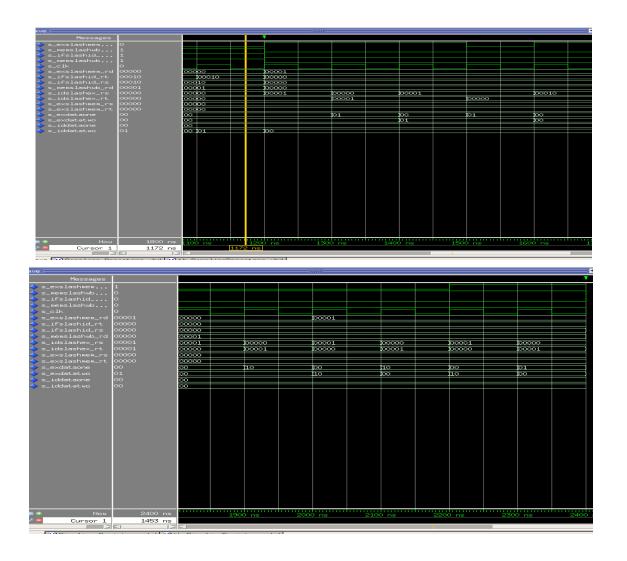
#### 4. MIPS Pipelined Processor

After putting in the pipelined registers we then created a Forwarding unit and Hazarding Unit which the test benches are listed shown below.

# Forwarding:

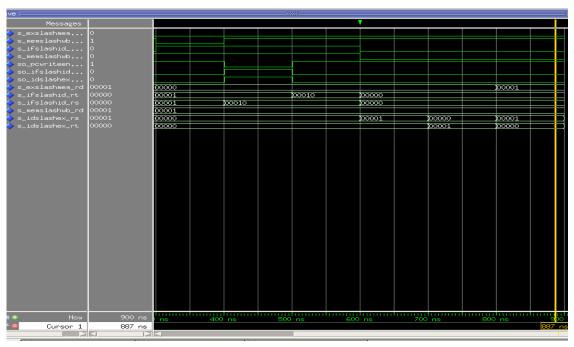


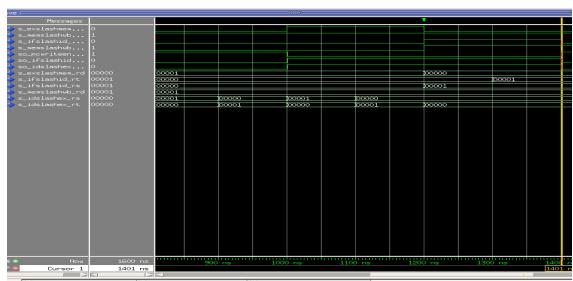
CprE 381 – Project Part C Group 9: Scott Connell, Brian Reber, Arjay Vander Velden



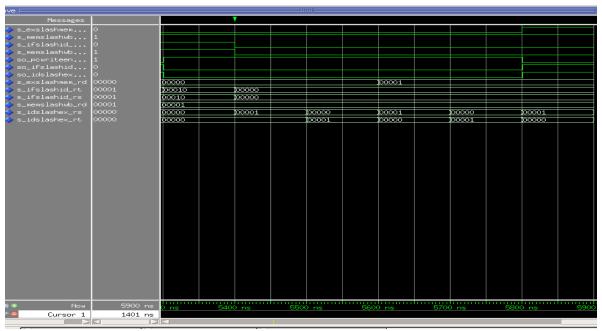
Hazard Detection:

 $\label{eq:cpreson} \textit{CprE 381} - \textit{Project Part C} \\ \textit{Group 9: Scott Connell, Brian Reber, Arjay Vander Velden}$ 



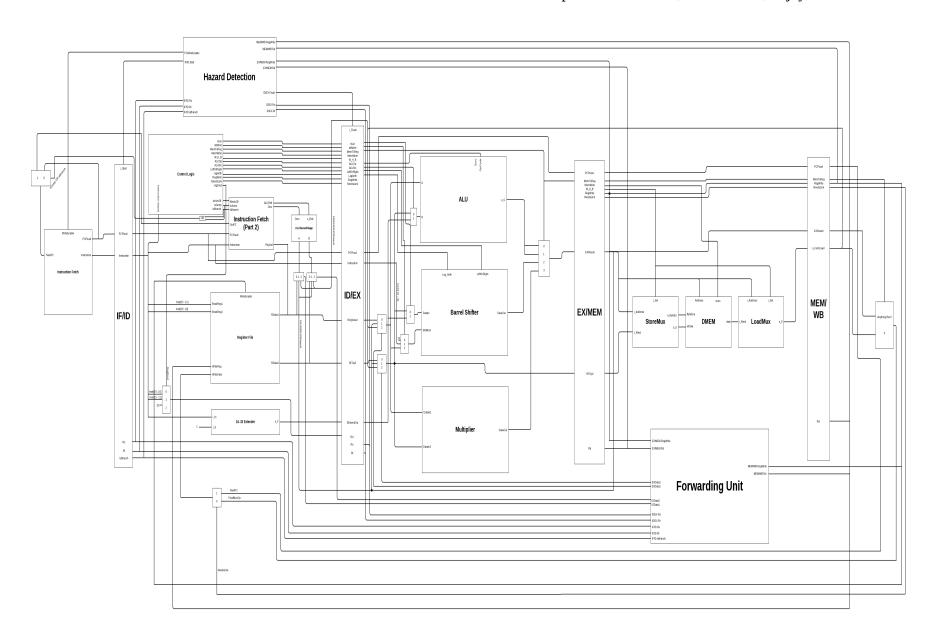


 $\label{eq:cpreson} \textit{CprE 381} - \textit{Project Part C} \\ \textit{Group 9: Scott Connell, Brian Reber, Arjay Vander Velden}$ 



Our Design is as follows below:

CprE 381 – Project Part C Group 9: Scott Connell, Brian Reber, Arjay Vander Velden



If you can not see our image check out the png in the zip file.

#### 5. <u>Testing</u>

- a) First we made and tested an application that uses every instruction **without** any data dependencies. It does not have any data dependencies because our code has a lot of no ops Our Code is in the test/asm folder and is called allinstructionsNoDependcies.s .
- b) Next we made and tested an application that uses every instruction **with** exhaustively the data dependencies of our processor. Our Code is in the test/asm folder and is called Allinstructions.s.
- c) Then we tested with our two sorting algorithms from Project Part B. Also located in the test/asm is the MergeSort.s and the Bubblesort.s.