[CprE 381] Computer Organization and Assembly-Level Programming, Fall 2018

Project B - Report

 Name(s)
 Yao Jiang Cheah / Alain Njipwo

 Section/Lab Time
 _____Thursday 4.10 - 6_____

Refer to the highlighted language in the <u>Project B</u> instruction for the context of the following questions.

a. [Part 1] Explain the conflict and how it relates to assumption that we will be making for forwarding and hazard detection (Hint: It has to do with the fact that we assume an instruction in the ID stage will get the data being written by the WB stage).

Give an example sequence of instructions that the processor we are designing will not be able to properly handle without additional logic or another change.

The conflict arises when the corresponding few instructions are dependent on the first instruction. This will cause many hazards including data hazards, and control hazards. This happens because the new instruction that is in the ID stage would try to read data from the register but without the value in the register being updated by the previous instruction that the new data is probably still in the EX, MEM, or WB stage. In order to resolve the data hazards, we would first implement a forwarding unit at the EX stage to forward EX/MEM ALU output or MEM/WB final output into the input of the ALU. Data hazard detection unit is needed to stall or flush the pipeline register because with some combination of instruction (lw follow by ALU instruction), a stalling is required because we will have to wait for the data to be computed somewhere in the pipeline before we can forward it using the forwarding unit.

Example:

- Required forwarding
 - o addi \$4, \$0, 1
 - o addi \$2, \$4, 2
 - o add \$3, \$0, \$4
- Also required hazard detection unit to stall and flush
 - lw \$8, 20(\$0)
 - o add \$9, \$8, \$8
- b. [Part. 4] Provide a description of a few test cases (at least one test case for each of the six instructions) and clear screenshots depicting your functioning test cases.
 - ADD
 - 0x01508020 -- add \$16, \$10, \$16

$0\lambda 01500020$ $uuu \oplus 10, 1$	$\varphi_{10}, \varphi_{10}$,							
/mipsprocessor_pipe/Clock	1								
/mipsprocessor_pipe/Reset	0								
/mipsprocessor_pipe/w_if_instruction	32'hAD300000	32'h0000	32h01508020	32'h00000000					
	32h00000000	32'h0000000)	32'h01508020	32'h00000000				
	32'h00000000	32'h0000000	j		32'h01508020	32h00000000			
	32'h00000000	32'h0000000)			32h01508020	3	2'h00000000	
	32'h00000000	32'h8D2A	32h0000000				13	2 ^h 01508020	
	32'd0	32'd0				32'd15	3	2'd0	
	32'd0	32'd5	1 32'd0				13	2'd15	
	32'd5	32'd5							
/mipsprocessor_pipe/inst_id_reg/registers(16)	32'd15	32'd10						(32'd15

• ADDI

0x21290004 -- addi \$9, \$9, 4

······································							
/mipsprocessor_pipe/Clock	1						
/mipsprocessor_pipe/Reset	0						
	32'h00000000	32'h0000000	32h21290004	32'h00000000			
	32'h2108FFFF	32'h0000000		32'h21290004	32'h00000000		
	32'h00000000	32'h0000000			32h21290004	32'h00000000	
+ /mipsprocessor_pipe/w_mem_instruction	32'h00000000	32'h0000000				32'h21290004	32'h00000000
	32'h00000000	32'hAD300000	32'h00000000				32'h21290004
	32'd0	32'd0				32'd24	, 32'd0
	32'd0	32'd20	32'd0				32'd24
+ /mipsprocessor_pipe/inst_id_reg/registers(9)	32'd24	32'd20					(32'd24
	32'd6	32'd6					

Register \$9 hold the value 20 and it's being add with immediate value 4. Therefore, the value updated to 24. We can also see how the value 24 is computed and pass to w_mem_ALU_out and to w_wb_mux_32 and into the register.

- LW
 - 0x8d2a0000 -- 1w \$10, 0(\$9)

- 🍲	/mipsprocessor_pipe/Clock	1									
- 🍐	/mipsprocessor_pipe/Reset	0								للسمع	
• 🔇	/mipsprocessor_pipe/w_if_instruction	32'h01508020	32ħ00000	000	32'h8D2A0000	32'h00000000					
• 🔸	/mipsprocessor_pipe/w_id_instruction	32'h00000000	32h00000	000		32'h8D2A0000	32'h00000000				
• 🔶	/mipsprocessor_pipe/w_ex_instruction	32'h00000000	32h00000	000			32h8D2A0000		32'h00000000		
• 🔷	/mipsprocessor_pipe/w_mem_instruction	32'h00000000	32ħ00000	000					32h8D2A0000	32ħ00000000	
	/mipsprocessor_pipe/w_wb_instruction	32'h00000000	32h00000	000						32h8D2A0000	
• 🔷	/mipsprocessor_pipe/w_mem_ALU_out	32'd0	32'd0						32'd20	32'd0	
• 🔷	/mipsprocessor_pipe/w_wb_mux32	32'd0	32'd0							32'd5	
• 🔸	/mipsprocessor_pipe/inst_id_reg/registers(9)	32'd20	32'd20								
u 🌖	/mipsprocessor_pipe/inst_mem_dmem/address	10'd0	10'd0					(10'd5	10'd0	
• 🔷	/mipsprocessor_pipe/inst_mem_dmem/mem(5)	32'd5	32'd5								
- 4	Iminsprocessor nine/inst id reg/registers(10)	32'd5	37/44								37/45

Register \$9 initially hold value of 20. Dividing by 4, so the memory address value that register 9 point to is 5. Then the value in memory 5 hold value of 5. It is then being read and store into register 10.

• SW

0xad300000 -- sw \$16, 0(\$9)

	and the second	N 1 1										
4	/mipsprocessor_pipe/Clock	1										
4	/mipsprocessor_pipe/Reset	0										
•	/mipsprocessor_pipe/w_if_instruction	32'h21290004	32'h00	000000	32'hAD300000	32'h00000000						
	/mipsprocessor_pipe/w_id_instruction	32'h00000000	32'h00	000000		32'hAD300000	32'h00000000					
	/mipsprocessor_pipe/w_ex_instruction	32'h00000000	32'h00	000000			32'hAD300000		32h00000000			
	/mipsprocessor_pipe/w_mem_instruction	32'h00000000	32'h00	000000					32hAD300000		32'h00000000	
H -	/mipsprocessor_pipe/w_wb_instruction	32'h00000000	32'h01	508020	32'h00000000						32'hAD300000	
	/mipsprocessor_pipe/w_mem_ALU_out	32'd0	32'd0						32'd20		(32'd0	
	/mipsprocessor_pipe/w_wb_mux32	32'd0	32'd15		32'd0						(32'd20	
	/mipsprocessor_pipe/inst_id_reg/registers(16)	32'd15	320	(32'd15								
•	/mipsprocessor_pipe/inst_id_reg/registers(9)	32'd20	32'd20									
	/mipsprocessor_pipe/inst_mem_dmem/address	10'd0	10'd0					(10'd5		(10'd0	
	/mipsprocessor_pipe/inst_mem_dmem/mem(5)	32'd15	32'd5							32'd15		

Register \$16 initially holds a value of 15. Register \$9 initially holds a value of 20. Dividing by 4, so the register \$9 direct to memory address 5. So the value of 15 in register \$16 is being saved into memory address 5.

• BEQ

Branch true: 0x10000004 -- beq \$zero, \$zero, test9 0x2008000c -- addi \$t0, \$zero, 12 0x00000000 0x00000000 0x00000000 0x20080014 -- test9: addi \$t0, \$zero, 20

/mipsprocessor_pipe/Clock	0												
/mipsprocessor_pipe/Reset	0												
	32'h2008000B	32'h 10000	004	32'h2008000	c 🕻	32'h200800:	4	32'h0000000	0			32'h2008000	B
	32'h00000000	32'h00000	000	32h100000	4	32'h0000000	00	32h200800	4	32'h0000000	0		
	32'h00000000	32'h00000	000			32h100000)4	32'h0000000	0	32'h200800	.4	32'h0000000	0
	32'h20080014	32'h00000	000					32h1000000	4	32'h0000000	0	32'h2008001	.4
	32'h00000000	32'h08000	034	32'h0000000	0					32h1000000	4	32'h0000000	0
/minsprocessor_nine/w_PCSrc	0												

Waveform above shown the branch instruction is implemented at the ID stage, which is when the instruction 0x2008000C was taken. Then the condition came to true so it branches to 0x20080014. The wire w_PCSrc is the branch combined condition that is the output from output of branch comparator AND branch_control.

Branch false: 0x11000003 -- beq \$t0, \$zero test11 0x20090032 -- addi \$t1, \$zero, 50 0x00000000 0x00000000 - test 11: // impaprocessor .ppe/_struction -/ impaprocessor .

In this case, the branch condition is not taken. Therefore, after stalling the isntruction 0x20080032 for one clock cycle, the program decided to compute it since the branching condition comes to false. In this case, due to data hazard, the branch condition w_PCSrc that we are looking at is the second clock cycle of instruction 0x20080032.

32'h00000000

32h1100000

32'h2009003

32h110000

32'h0000000

J										
0x08000034 j test8										
0x2009000a add \$t1, \$zero	, 10									
0x0000000	, 									
0x0000000										
0x0000000										
0x00000000 test8:										
0x0000000										
0x10000004 beg \$zero, \$ze	ro, test9									
/mipsprocessor_pipe/Clock	0									
/mipsprocessor_pipe/Reset	0									
+ /mipsprocessor_pipe/w_if_instruction	32'h20080014	32'h08000034	32h2009000	A	32'h000000	00			32h100000	14
	32'h00000000	32'h00000000	132'h0800003	4	132'h000000	00				
	32'h10000004	32'h00000000			32'h080000	34	132h000000	0		
	32'h00000000	32'h00000000					32h0800003	4	32'h0000000	0
+	32'h00000000	32'h00000000							32'h0800003	4
/mipsprocessor_pipe/w_id_jump	0									
+	32'h00000000	32'h00000000	32'h0000000	0	1 32'h000000	00				
/mipsprocessor_pipe/w_id_nextPC	32'h00000000	32'h000000BC	32'h0000000	0	32'h000000	00	32'h000000)4	32'h000000	8

In the above case, the instruction jump at the ID stage, which is the same time when instruction 0x2009000a is being read into IF stage. Then the instruction 0x2009000a is being flushed and the program jump to the test8 line which is the 2^{nd} last 0x00000000 instruction before beq instruction. After two clock cycles, it officially enters the 0x10000004 instruction.

c. [Part 5] Implement ID stage branch resolution and provide a legible simulation-screenshot of a takenbranch instruction correctly executing.

Since we have already implemented ID stage branch resolution in the previous part, it will be the same:



Waveform above shown the branch instruction is implemented at the ID stage, which is when the instruction 0x2008000C was taken. Then the condition came to true so it branches to 0x20080014. The wire w_PCSrc is the branch combined condition that is the output from output of branch comparator AND branch_control.

Branch false: 0x11000003 beq \$t0 0x20090032 addi \$t: 0x00000000 0x00000000	, \$zero t l, \$zero,	est11 50										
000000000000000000000000000000000000												
/mipsprocessor_pipe/Clock	1											
/mipsprocessor_pipe/Reset	0											
	32h0000000	32h11000003	C 32h2009003	2		32700000000						
	32'h00000000	32m20080028	(32h1100000	3		132h200900\$2	32'h000000	0				
	32h00000000	32h00000000	32h2008002	8 (3	32'h00000000	32h11000003	32h2009003	2	32'h0000000	0	العمديين	
	32'h00000000	32h00000000		10	32h20080028	32h0000000	32'h1100000	3	132h2009003	2	32'h0000000	0
	32'h00000000	32'h00000000				32120080028	32'h0000000	0	32h1100000	3	132h2009003	2
/mipsprocessor_pipe/w_PCSrc	0											

In this case, the branch condition is not taken. Therefore, after stalling the isntruction 0x20080032 for one clock cycle, the program decided to compute it since the branching condition comes to false. In this case, due to data hazard, the branch condition w_PCSrc that we are looking at is the second clock cycle of instruction 0x20080032.

- d. [Part 6a] Implement a forwarding unit (using VHDL) to support the following data dependent cases. Give simulation screenshots of correct forwarding for each case, make sure to provide an explanation of the instructions you ran to show the below hazards:
 - i) ALU producer to ALU consumer at distance 1 (e.g. ADD \$1, \$2, \$3; ADD \$4, \$1, \$2)

0x20080001 addi \$8, \$zero, 1
0x00084820 add \$9, \$zero, \$8
0x01205020 add \$10, \$9, \$zero
/mipsprocessor_pipe/Clock

/mipsprocessor_pipe/Clock	0														
/mipsprocessor_pipe/Reset	0														
	32h00085020	32h2008000	1	32'h0008482	0	32'h0120502	0	32'h000000	0			32h2008000)4	32'h2009FFF	F
	32h2009FFFF	32'h0000000	0	32h2008000	1	32h0008482	0	32h0120502	20	32'h000000	0			32'h2008000	4
	32h20080004	32h000000	0			32h2008000	1	32h0008482	20	32'h012050	0	32h000000	00		
	32h0000000	32h000000	0					32h2008000)1	32'h000848	20	32h012050	20	32'h0000000	0
mipsprocessor_pipe/w_wb_instruction	32h0000000	32'h0000000	0							32'h2008000	1	32h000848	20	32'h0120502	0
/mipsprocessor_pipe/inst_id_reg/registers(10)	32'd1	32'd0													32'd1
/mipsprocessor_pipe/inst_id_reg/registers(9)	32'd1	32'd0											32'd1	```	
/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd1	32'd0								(32'd1				

As we can see, the register \$9 is updated with the value of register \$8 (value 1) even with distance 1 ALU data hazard.

ii) ALU producer to ALU consumer at distance 2 (e.g. ADD \$1, \$2, \$3; <INST>; ADD \$4, \$1, \$2)
 0x20080004 -- addi \$8, \$zero, 4
 0x2009ffff -- addi \$t9, \$zero, -1

0x20091111 -- addi \$19, \$2ero, -1 0x00085020 -- add \$10, \$zero, \$8

ا	/mipsprocessor_pipe/Clock	0														
- 🎸	/mipsprocessor_pipe/Reset	0														
	/mipsprocessor_pipe/w_if_instruction	32'h20080005	32h2008	004	32h2009FFF	F	32'h0008502	0	32h0120582	0	32h000000	0			32h2008000	16
	/mipsprocessor_pipe/w_id_instruction	32'h20080006	32'h00000	0000	32'h2008000	4	32'h2009FFF	F	3210008502	0	32h012058	20	32h000000	00		
∎	/mipsprocessor_pipe/w_ex_instruction	32'h00000000	32'h00000	000			32h2008000	4	32h2009FFF	F	32h000850	20	32h012058	20	32'h0000000	00
.	/mipsprocessor_pipe/w_mem_instruction	32'h00000000	32'h0120	020	32'h0000000	0			32h2008000	4	32h2009FF	F	32h000850	20	32/h0120583	20
±	/mipsprocessor_pipe/w_wb_instruction	32'h01205820	32'h0008+	820	32'h012050	0	32'h0000000	0			32 h 200800)4	32h2009FFF	F	32h0008503	20
	/mipsprocessor_pipe/inst_id_reg/registers(11)	-32'd1	32'd0													
₽-�	/mipsprocessor_pipe/inst_id_reg/registers(10)	32'd4	32'd0			32'd1										32'd4
•	/mipsprocessor_pipe/inst_id_reg/registers(9)	-32'd1	32'd0	32'd1										-32'd1		
+	/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd4	32'd1									1 32'd4				

As we can see, the register \$10 is updated with the value of register \$8 (value 4) even with distance 2 ALU data hazard.

 iii) Load producer to ALU consumer distance 2 (e.g. LW \$1, 0(\$10); <INST>; ADD \$5, \$1, \$r2) 0x8c080018 -- 1w \$8, 24(\$zero) 0x00000000 0x01084820 -- add \$9, \$8, \$8

 mapprocessor_pape/list strangerocessor_pape/list strangeroceso

32'di

The immediate value 24 corresponds to address 6 in the memory. From the waveform above, the value of memory 6 is 6 and it is being updated to register \$8. Then the value in register \$8 is being added to itself and save to register \$9.

10'd6

10'd0

4	/mipsprocessor_pipe/Clock	1															
4	/mipsprocessor_pipe/Reset	0															
H -	/mipsprocessor_pipe/w_if_instruction	32'h2009000A	32h11000003		32h0000000	0										32h080000	34
	/mipsprocessor_pipe/w_id_instruction	32'h08000034	32100000000	10	32h1100000	3	3211000000	00									
B - (/mipsprocessor_pipe/w_ex_instruction	32'h00000000	32h20080023		32110000000	0	32h110000)3	32/h000000	0							
B - (/mipsprocessor_pipe/w_mem_instruction	32'h00000000	32'h00000000)(32h200800	3	32h000000	00	32h110000	03	32'h000000	00					
	/mipsprocessor_pipe/w_wb_instruction	32'h00000000	32'h00000000				32h200800	23	32/h000000	00	32'h110000	03	3216000000	00			
H -	/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd35 🤇	32'd0				(32'd35									
4	/mipsprocessor_pipe/w_PCSrc	0															
	1		.1				1		111		. 11				• •	 1	

The two simulations above are the same instructions but it will not fit all outputs so we included two screenshots. As we can see, the branching condition still works whereas it comes to false and clock at 0x00000000 for 6 cycles before entering the jump instruction because it took the value 35 (new value of register \$8) instead of the value 0 (old value of register \$8) initially that was saved using the instruction 0x20080000.

- e. [Part 6b] Implement a hazard detection unit (using VHDL) to support the following data dependent cases. Give simulation screenshots of correct forwarding for the cases described in the lab manual, provide an explanation.
- f.

Hazard detection and forwarding unit test (HDandFUtest):

Test 1: ALU Producer to ALU Consumer EX/MEM to EX

/mipsprocessor_pipe/Clock	0												
/mipsprocessor_pipe/Reset	0												
/mipsprocessor_pipe/w_if_instruction	32'h00085020	32'h00084	4820	32'h0120502	0	32'h0000000	0			32'h2008000	4	32'h2009FFF	
/mipsprocessor_pipe/w_id_instruction	32'h2009FFFF	32'h20080	0001	32'h0008482	0	32'h0120502	0	32'h0000000	0			32'h2008000	4
mipsprocessor_pipe/w_ex_instruction	32'h20080004	32'h00000	0000	32'h2008000	1	32'h0008482	0	32h0120502	0	32'h0000000	0		
/mipsprocessor_pipe/w_mem_instruction	32'h00000000	32'h00000	0000			32'h2008000	1	32h0008482	0	32h0120502	0	32'h0000000)
/mipsprocessor_pipe/w_wb_instruction	32'h00000000	32'h00000	0000					32h2008000	1	32h0008482	0	32'h0120502)
/mipsprocessor_pipe/inst_id_reg/registers(11)	32'd0	32'd0											
/mipsprocessor_pipe/inst_id_reg/registers(10)	32'd1	32'd0											32'd1
	32'd1	32'd0									32'd1		
/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd1	32'd0							32'd1				

Value 1 is being forwarded from register \$8 at distance 1.

Test2: ALU Producer to ALU Consumer MEM/WB to EX

	/mipsprocessor_pipe/Clock	1																	
	/mipsprocessor_pipe/Reset	0																	
- K		32'h0 1084820	32'	32h20080	004	32h2009F	FFF	32100085	020	32'h01205	820	32h00000	000			32h20080	006	32h20080	005
- K		32h20080005	32'h00	000000		32'h20080	004	32h2009F	FF	32'h00085	020	32h01205	820	32'h00000	000			32h20080	006
- K		32'h20080006	32'	32'h00000	000			32h20080	004	32'h2009E	166	32h00085	020	32'h01205	820	32h00000	000		
Ľ		32'h00000000	32'	32'h01205	020	32'h00000	000			32'h20080	004	32h2009F	FFF	32'h00085	020	32h01205	820	32h00000	000
Ľ		32'h00000000	32'	32'h00084	820	32h01205	020	32700000	000			32h20080	004	32h2009E	FFF	32h00085	020	32h01205	820
Ľ		-32'd1	32'd0																-32'd1
Ľ		32'd4	32'd0				32'd1										32'd4		
Ľ	Impsprocessor_pipe/inst_id_reg/registers(9)	-32'd1	32'd0		32'd1										-32'd1				الكف
- 8	Imperior cases on pipe linet in registers (8)	37/44	20/41										22/44						

Value 4 is being forwarded into register \$10 and the value -1 is being forwarded to register \$11.

Test3: ALU Producer to ALU Consumer Precedence Test

- 🚰 🗸	Msgs															
/mipsprocessor_pipe/Clock	1															
/mipsprocessor_pipe/Reset	0															
	32'h00085020		32h20080	006	32'h20080	005	32'h01084	820	32h00000	000			32'h20080	00A	32'h20000	800
	32'h00004820	32	100000000		32'h20080	006	32'h20080	005	32ĥ01084	820	32'h00000	000			32h20080	00A
	32'h2000008		32'h00000	000			32'h20080	006	32h20080	005	32'h01084	820	32'h00000	000		
	32'h2008000A		32ĥ01205	820	32'h00000	000			32 h 20080	006	32'h20080	005	32'h01084	320	32'h00000	000
	32'h00000000		32'h00085	020	32'h01205	820	32'h00000	000			32'h20080	006	32'h20080	005	32'h01084	820
	-32'd1	32	d0			-32'd1										
	32'd4	32	d1	32'd4												
	32'd10	-32	d1													32'd10
	32'd5	32	d4									32'd6		32'd5		

Value 5 is being forwarded for adding with itself.

Test4: Zero Register Forwarding

- 💫 🗸	Msgs																
/mipsprocessor_pipe/Clock	1																
/mipsprocessor_pipe/Reset	0																
Mipsprocessor_pipe/w_if_instruction	32'hAD090000	32h200	8000A	32ħ20000	008	32h00004	320	32h00085	020	32ħ00000	000			32ħ20080	004	32h2009F	-9C
	32'h2009FF9C	32h000	00000	32 h 20080	00A	32ħ20000	800	32100004	320	32h00085	020	327600000	000			32h20080	004
mipsprocessor_pipe/w_ex_instruction	32h20080004	32h000	00000			32h20080	AOC	32h20000	008	32h00004	820	32h00085	020	32ħ00000	000		
mipsprocessor_pipe/w_mem_instruction	32'h00000000	32h010	84820	32ħ00000	000			32h20080	AOC	32ħ20000	800	32h00004	820	32ħ00085	020	32'h00000	000
	32'h00000000	32h200	80005	32h01084	820	32h00000	000			32h20080	00A	32h20000	008	32h00004	820	32h00085	020
mipsprocessor_pipe/inst_id_reg/registers(11)	-32'd1	-32'd1															
#	32'd10	32'd4															32'd10
Mipsprocessor_pipe/inst_id_reg/registers(9)	32'd0	-32'd1			32'd10										32'd0		
/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd10	32'd6	32'd5								32'd10						

Zero register is not being forwarded.

Test5: Forwarding to a SW

	6															
	/mipsprocessor_pipe/Clock	1														
	/mipsprocessor_pipe/Reset	0														
÷	/mipsprocessor_pipe/w_if_instruction	32'h00000000	32'h	20080004	32h2009F	-9C	32'hAD090	000	32'h00000	000			32h20080	008	32'h8D090	000 (
٠	/mipsprocessor_pipe/w_id_instruction	32'h2008000C	32'h	00000000	32'h20080	004	32'h2009F	-9C	32'hAD090	000	32'h00000	000			32'h20080	008 X
÷	/mipsprocessor_pipe/w_ex_instruction	32'h8D090000	32'h	00000000			32h20080	004	32h2009F	F9C	32'hAD090	000	32'h00000	000		X
÷		32'h20080008	32'h	00085020	32'h00000	000			32h20080	004	32'h2009F	F9C	32'hAD090	000	32'h00000	000
÷	/mipsprocessor_pipe/w_wb_instruction	32'h00000000	32'h	00004820	32'h00085	020	32'h00000	000			32'h20080	004	32'h2009F	F9C	32'hAD090	000 X
÷	/mipsprocessor_pipe/inst_id_reg/registers(11)	-32'd1	-320	1												
٠	/mipsprocessor_pipe/inst_id_reg/registers(10)	32'd10	32'd	4		32'd10										
÷	/mipsprocessor_pipe/inst_id_reg/registers(9)	-32'd100		32'd0										-32'd100		
÷	/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd4	32'd	10								32'd4				
	/mipsprocessor_pipe/inst_mem_dmem/mem(1)	-32'd100	32'd	1										-32'd100		

The value -100 is being forwarded for sw, mem(1) is updated to value -100.

Test6: Forwarding to a LW

U																			
/mipsprocessor_pipe/Clock	1																		
/mipsprocessor_pipe/Reset	0																		
/mipsprocessor_pipe/w_if_instruction	32h00000000	32h2008	0008	32h8D090	0000	32'h2008	000C	32'h0000	0000	32h8D0A	0000	32'h0000	000			32'h2008	0000	32100000	000
	32h0000000	32'h0000000	0	32h20080	008	32'h8D09	0000	32'h2008	000C	32h0000	000	32'h8D0A	0000	32'h0000	0000			32h20080	000
	32h20080000	32h0000	0000			32'h2008	0008	32/h8D09	0000	32h2008	00C	32h0000	000	32'h8D04	0000	32'h0000	0000		
	32h0000000	32hAD09	0000	32700000	000			32h2008	0003	32h8D09	0000	32h2008	00C	32'h0000	0000	32'h8D0A	0000	321100000	0000
Herein - Antipage -	32h0000000	32h2009	F9C	32hAD09	0000	32'h0000	0000			32h2008	008	32h8D09	0000	32'h2008	000C	32'h0000	0000	32768D0A	0000
	-32'd1	-32'd1																	
	32'd3	32'd10																	32'd3
/mipsprocessor_pipe/inst_id_reg/registers(9)	32'd2	32'd0	-32'd100										32'd2						
/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd12	32'd4									32'd8				32'd12				
/mipsprocessor_pipe/inst_mem_dmem/mem(1)	-32'd100	32'd1	-32'd100																
/mipsprocessor_pipe/inst_mem_dmem/mem(2)	32'd2	32'd2																	
/mipsprocessor_pipe/inst_mem_dmem/mem(3)	32'd3	32'd3																	

Value 8 is being forwarded to lw as a loading address rs.

Test7: Forwarding to Branch

4	/mipsprocessor_pipe/Clock	1																	
4	/mipsprocessor_pipe/Reset	0																	
± -	/mipsprocessor_pipe/w_if_instruction	32'h20080000	32h20080000	32'h0000	000					32h2008	0023	32'h0000	¢000	32h1100	0003	32'h000	00000		
H -	/mipsprocessor_pipe/w_id_instruction	32'h00000000	32'h00000000	32'h2008	000	32 `h0000 0	000					32'h2008	0023	32h0000	0000	32'h110	00003	32h00	000000
B -4	/mipsprocessor_pipe/w_ex_instruction	32'h00000000	32'h00000000			32h20080	000 32	'h00000	000					32h2008	0023	32'h000	00000	32h1	1000003
±-<	/mipsprocessor_pipe/w_mem_instruction	32'h8D0A0000	32'h8D0A0000	32'h0000	000		(32	(h2008¢	000	3210000	0000					32'h200	80023	32h00	000000
±	/mipsprocessor_pipe/w_wb_instruction	32'h00000000	32'h00000000	32'h8D0A	0000	327h00000	000			32h2008	0000	32'h0000	¢000					32h20	080023
	/mipsprocessor_pipe/inst_id_reg/registers(11)	-32'd1	-32'd1																
H	/mipsprocessor_pipe/inst_id_reg/registers(10)	32'd10	32'd10		32'd3														
±	/mipsprocessor_pipe/inst_id_reg/registers(9)	32'd2	32'd2																
.	/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd12	32'd12								32'd0								32'd35
	/mipsprocessor_pipe/Clock	1																	
	/mipsprocessor_pipe/Reset	0																	
	/mipsprocessor_pipe/w_if_instruction	32h20080000	32ħ00000	000						32'h08	000034	32'h20090	100A 3	2'h0000000)	1	32'h 10000	004 3	2'h2008000C
	/mipsprocessor_pipe/w_id_instruction	32h0000000	32h00000	000								32'h08000	034 3	2'h0000000)			3	2'h10000004
	/mipsprocessor_pipe/w_ex_instruction	32ħ00000000	32h1	32'h0000000	0								3	2'h0800003	1 32'h00	00000			
	/mipsprocessor_pipe/w_mem_instruction	32h8D0A0000	32ħ0	32h1100000	3 🕻 32'h000	00000									32'h08	000034	32'h00000	000	
	/mipsprocessor_pipe/w_wb_instruction	32ħ00000000	32h2	32'h0000000	0 32'h110	00003 🕻 3	32'h0000000	0								X	32'h08000	034 3	2'h00000000
	/mipsprocessor_pipe/inst_id_reg/registers(11	.) -32'd1	-32'd1																
	/mipsprocessor_pipe/inst_id_reg/registers(10) 32'd10	32'd3																
	/mipsprocessor_pipe/inst_id_reg/registers(9)	32'd2	32'd2																
	/mipsprocessor_pipe/inst_id_reg/registers(3)	32'd12) 32'd35																
			_	-															

Two waveforms above are continuous of each other. The code branched at ID stage.

Test8: Jump Followed by an Add

· · · · · · · · · · · · · · · · · · ·													
/mipsprocessor_pipe/Clock 0	D												
/mipsprocessor_pipe/Reset	D												
	32'h20080014	32'h08000034	321	h2009000A	, i i i i i i i i i i i i i i i i i i i	32'h0000000	0			32'h1000000	4	32'h2008000	С
mipsprocessor_pipe/w_id_instruction 33	32'h00000000	h00000000	321	h08000034	1	32'h0000000	0					32'h 1000000	4
mipsprocessor_pipe/w_ex_instruction 33	32'h10000004	h00000000			X	32'h080000	4	32'h0000000	0				
	32'h00000000	h00000000						32'h0800003	4	32'h0000000	0		
	32'h00000000	h00000000								32'h0800003	4	32'h0000000	0
mipsprocessor_pipe/inst_id_reg/registers(11) -3	-32'd1 -3	2'd1											
mipsprocessor_pipe/inst_id_reg/registers(10) 33	32'd3	'd3											
	32'd2	'd2											

The add statement after jump was being flused therefore it is not computed.

Test9: Branch Taken then Add

/mipsprocessor_pipe/Clock	1														
/mipsprocessor_pipe/Reset	0														
	32'h00000000	32'h100000	04	32h200800	С	32h200800	4	32h000000	0			32h200800	0B	32'h000000	0
	32'h0000000	32'h000000	00	32h100000)4	32h000000	0	32h200800	4	32'h0000000	0			32h2008000	96
	32'h2008000B	32'h000000	00			32h100000	14	1 32'h0000000	0	32h200800	4	32h000000	00		
	32'h00000000	32'h000000	00					32h1000000	4	32'h0000000	0	32h200800	14	32'h0000000	00
	32'h00000000	32'h080000	34	32'h000000	00					32h1000000	4	32h000000	00	32'h200800	14
	-32'd1	-32'd1													
	32'd3	32'd3													
	32'd2	32'd2													
	32'd20	32'd35													32'd20

Branch is taken at ID stage so the instruction addi is being flushed.

Test10: Branch Not Taken then add

/mipsprocessor_pipe/Clock	1																					
/mipsprocessor_pipe/Reset	0																					
/mipsprocessor_pipe/w_if_instruction	32'h00000000	32h20	08000B	32'h000	00000					32h11	000004	32ħ01	84820	32'h000	00000							
/mipsprocessor_pipe/w_id_instruction	32'h00000000	32ħ0000	0000	32h200	3000B	32h000	00000					32h11	00004	32'h010	84820	32'h00	00000					
	32'h00000000	132'h00	000000			32h200	8000B	32h00	00000					32h110	00004	32'h01	84820	32'h00	00000			
	32'h00000000	[32'h20	080014	32'h000	00000			32h20	8000B	32h00	00000					32h11	00004	32h01	84820	32'h00	00000	
	32h0000000	132'h00	00000	32'h20	80014	32'n000	00000			32h20	80008	32h00	00000					32'h11	00004	32'h01	84820	(32'h000
/mipsprocessor_pipe/inst_id_reg/registers(11)	-32'd1	-32'd1																				
	32'd3	32'd3																				
	32'd22	32'd2																			32'd22	
	32'd11	32'd35			32'd20						32'd11											

Branch not taken so the instruction add is not being flushed at its ID stage.

Test11: Add followed by a branch

/mipsprocessor_pipe/Clock	0													
/mipsprocessor_pipe/Reset	0													
mipsprocessor_pipe/w_if_instruction	32h0000000	32h20080	32'h00000000)		32h20080	32h11000	32'h20090032		32h0000000				
mipsprocessor_pipe/w_id_instruction	32h0000000	32'h00000000	32'h20080	32/h00000000			32h20080	32h11000003		32h20090	32700000000			
mipsprocessor_pipe/w_ex_instruction	32h0000000	32'h00000000		32h20080	32h00000000			32'h20080	32h00000	32h11000	32h20090	32h00000000		
	32h0000000	32'h00000000			32h20080	32h00000000			32h20080	32h00000	32h11000	32h20090	32/h00000000	
	32'h00000000	32'H00000000				32h20080	32'h00000000)		32h20080	32h00000	32h11000	32h20090	32h00
/mipsprocessor_pipe/inst_id_reg/registers	(11) -32'd1	-32d1												
/mipsprocessor_pipe/inst_id_reg/registers	(10) 32'd3	32'd3												
/mipsprocessor_pipe/inst_id_reg/registers	(9) 32'd50	32'd22											32/d50)
mipsprocessor_pipe/inst_id_reg/registers	(8) 32'd40	32'd11				(32'd0) 32'd40				

The addi operation stall the beq (at ID/EX) for one cycle to wait for the result to be done before forwarding it to the beq.

Test12: Lw to Add distance 1

hinterrocentor pipe/Clock	4																					
/mpsprocessor_pipe/clock	1																					
/mipsprocessor_pipe/Reset	0																					
/mipsprocessor_pipe/w_if_instruction	32'h00000000	32'h0000402	32h0	0000000					32h8C0	80014	32h010	84820	32h000	00000							32h000	04020
mipsprocessor_pipe/w_id_instruction	32'h00004020	32'h0000000	32h0	0004020	32'h000	00000					32h8C0	80014	32h010	34820			32 `h 000	00000				
/mipsprocessor_pipe/w_ex_instruction	32'h00000000	32'h0000000			32'h000	04020	32h000	00000					32h8C0	80014	32'h000	00000	32h010	34820	32h000	00000		
/mipsprocessor_pipe/w_mem_instruction	32'h00000000	32'h0000000					32h000	04020	32'h000	00000					32'h8C0	80014	32h000	00000	32'h010	34820	32'h000	00000
mipsprocessor_pipe/w_wb_instruction	32'h00000000	32'h0000000							32h000	04020	32h000	00000					(32h8C0	80014	32h000	00000	32'h010	84820
/mipsprocessor_pipe/inst_id_reg/registers(11)	-32'd1	-32'd1																				
/mipsprocessor_pipe/inst_id_reg/registers(10)	32'd3	32'd3																				
mipsprocessor_pipe/inst_id_reg/registers(9)	32'd10	32'd50																				32'd10
/mipsprocessor_pipe/inst_id_reg/registers(8)	32'd5	32'd40								32'd0								32'd5				
mipsprocessor_pipe/inst_mem_dmem/mem(1)	-32'd100	-32'd100																				
	32'd2	32'd2																				
/mipsprocessor_pipe/inst_mem_dmem/mem(3)	32'd3	32'd3																				
/mipsprocessor_pipe/inst_mem_dmem/mem(4)	32'd4	32'd4																		_		
/mipsprocessor_pipe/inst_mem_dmem/mem(5)	32'd5	32'd5																				
	32'd6	32'd6																				

The LW stall the add for one cycle (at ID/EX) to wait for the result to be done before forwarding it to the add.

Test13: Lw to Add distance 2

/mipsprocessor_pipe/Clock	0						1												
/mipsprocessor_pipe/Reset	0																		
	32h0000000	132h00004020	32h00000000			32h8C	080018	32h000	00000	1 32 h0 10	34820	32h000	00000						
Mipsprocessor_pipe/w_id_instruction	32h0000000	32h00000000	32h00004020	32h00000000				32h8C0	80018	32'h000	00000	32h010	84820	32'H000	00000				
	32ħ00000000	32h00000000		32h00004020	32/h0000000	00				32'h8C0	80018	32h000	00000	32'h010	84820	3276000	00000		
Mipsprocessor_pipe/w_mem_instructio	n 32h0000000	32h0000000			32/h0000402	20 (32'h000	00000					032h8C0	80018	32/6000	00000	32h010	34820	32h000	00000
	32h01084820	(32h01084820	32'h00000000			32h000	004020	32'h000	00000					32'h8C0	80018	32ħ000	00000	32'h010	84820
mipsprocessor_pipe/inst_id_reg/registered	ers(11) -32'd1	-32'd1																	
mipsprocessor_pipe/inst_id_reg/registered	ers(10) 32'd3	32'd3																	
mipsprocessor_pipe/inst_id_reg/registered	ers(9) 32'd12	32'd50 32'd10																	32612
Mipsprocessor_pipe/inst_id_reg/registered	ers(8) 32'd6	32'd5					32'd0								32'd6				
	nem(1) -32'd100	-32'd100																	
Mipsprocessor_pipe/inst_mem_dmem/r	nem(2) 32'd2	32'd2																	
	nem(3) 32'd3	32d3																	
Mipsprocessor_pipe/inst_mem_dmem/r	1em(4) 32'd4	32'd4																	
	nem(5) 32'd5	32'd5																	
	1em(6) 32'd6	32'd6																	

The lw flushed and stalled the add instruction for one clock cycle at the IF stage then only compute it.

 g. [Part 6c] Connect your forwarding and hazard detection units to your pipelined processor and provide a simulation screenshot showing that your pipeline correctly executes the given test program.
 From the simulation of "Summation test":



]	Memory data at the end of simulation (decimal):															
	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	55	45	36	28	21	15	10
	1	6	3	1	10											

Extra Credit (5 points) (This is not required):

- Explain the potential impact that the falling edge trigger register fill has on the critical path for the • processor we are designing (2.5 points). The memory is also falling edge trigger so if the output value from register is forwarded to the memory data input, it may cause a problem.
- Implement a fix (in VHDL) to the issue we have caused that will allow the register file to continue • to write on the rising edge. Draw (or screenshot) the change that you have made and explain how it solves the potential issue.