



Architecture Project

Team Number: 15

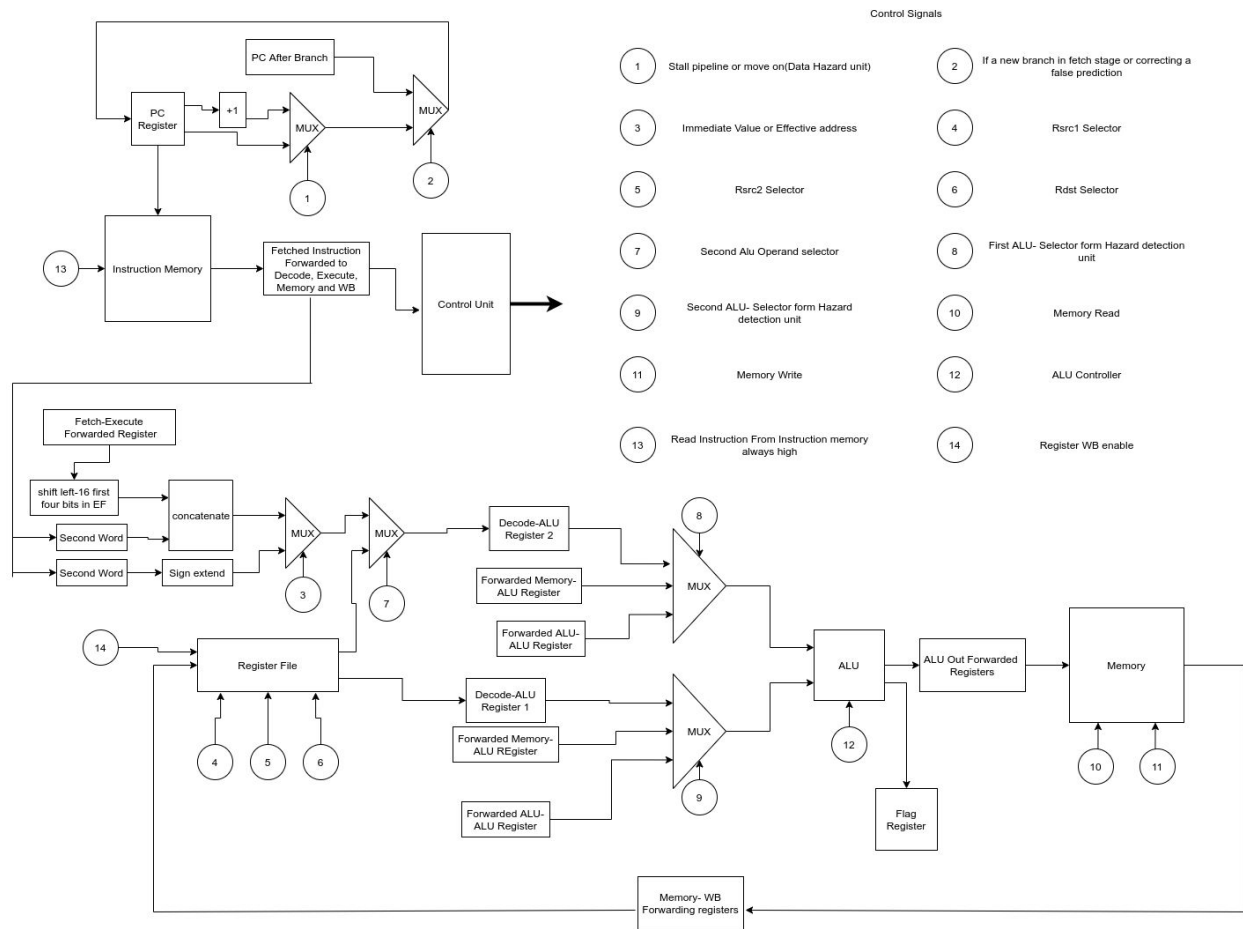
Members:

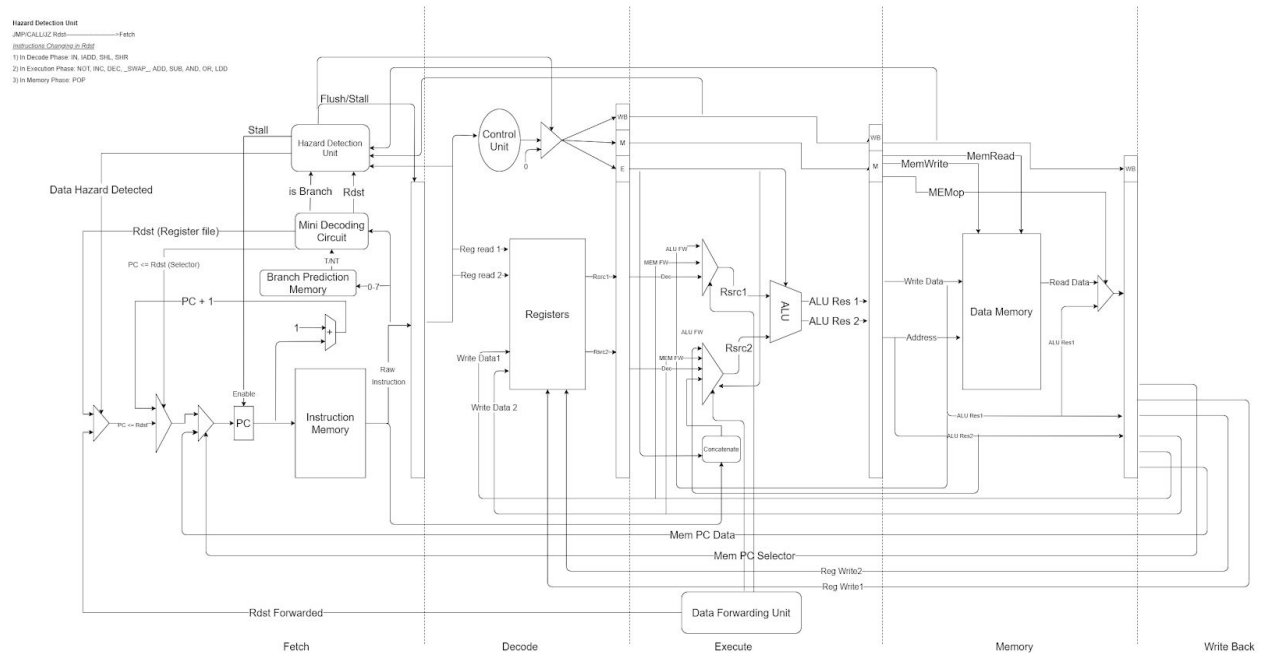
Name	Section	B.N
Ayman Abdelnaby	1	14
Khaled Moataz	1	18
Kareem Osama	2	5
Muhammad Sayed	2	15

Instruction Format:

2-Operands (0)	SHL (000)	5'X		Rdst (3-bits)	4'X	(2nd word) Imm	
	SHR (001)						
	SWAP (010)	Rsrc1 (3-bits)	2'X				Rsrc2 (3-bits)
	IADD (011)				(2nd word) Imm		
	ADD (100)						
	SUB (101)						
	AND (110)						
	OR (111)						
1-Operand (100)	ALU (100)	NOT(00)	1'X	Rdst (3-bits)	4'X		
		INC(01)					
		DEC(10)					
	OUT (101)	3'X					
	IN (110)						
Memory (101)	Stack (00)	POP(0)	3'X	Rdst (3-bits)	4'X		(2nd word) Imm
		PUSH(1)					
	Imm (01)	LDM (0)			EA[19-16]		(2nd word) EA[15-0]
	EA (10)	LDD (0)					
		STD (1)					
Branch (110)	JMP (0)	CALL (001)	2'X	Rdst (3-bits)	4'X		
		JMP (010)					
		JZ (100)					
	RET (1)	RET (01)	10'X				
		RTI (10)					
NOP (1110)	13'X						
INT/RST (1111)							

Schematic Diagram:





Control Signals:

Instruction	ALU 2nd Op: Reg(0), other(1)	Imm(0), EA(1)	Branch	Rsrc1 enable	Rsrc2 enable	Rdst WB	Memory Read	Memory Write	M(0)/A LU(1) WB	ALU Controller
SHL	1	0	0	1	0	1	0	0	1	0100
SHR	1	0	0	1	0	1	0	0	1	0101
SWAP	0	X	0	1	1	1	0	0	1	0001
IADD	1	0	0	1	0	1	0	0	1	1000
ADD	0	X	0	1	1	1	0	0	1	1000
SUB	0	X	0	1	1	1	0	0	1	1001
AND	0	X	0	1	1	1	0	0	1	0110
OR	0	X	0	1	1	1	0	0	1	0111
NOP	X	X	0	0	0	0	0	0	X	0000
NOT	0	X	0	1	0	1	0	0	1	0010
INC	0	X	0	1	0	1	0	0	1	1010
DEC	0	X	0	1	0	1	0	0	1	1011
OUT	0	X	0	1	0	0	0	0	X	0000
IN	0	X	0	0	0	1	0	0	X	0000

PUSH	0	X	0	1	0	0	0	1	X	1001
POP	0	X	0	0	0	1	1	0	0	1000
LDM	1	0	0	0	0	1	0	0	X	0001
LDD	1	1	0	0	0	1	1	0	0	0000
STD	1	1	0	1	0	0	0	1	X	0000
CALL	0	X	1	1	0	0	0	1	1	1001
JMP	X	X	1	1	0	0	0	0	X	0000
JZ	X	X	1	1	0	0	0	0	X	0011
RET	X	X	1	0	0	0	1	0	1	1000
RTI	X	X	1	0	0	0	1	0	1	1000

- Reset Signal: sets PC = 1, resets all control signals and all forwarding registers
- Interrupt Signal: sets PC = 2, 3 , save the flag register
- Instruction Memory Read Signal: always 1
- Port In Signal: 0 except In instruction 1
- Port out Signal: 0 except out instruction 1

Pipeline Stages Design:

Pipeline Registers Details:

Register	Size (bits)	Inputs
IF/ID	81	[15-0] : Instruction
		[47-16] : Incremented PC
		[48] : JZ initial prediction
		[80-49] : IN.value
ID/EX	138	[15-0] : Current instruction (from IF/ID) anded with 0x000F (used in EA calculation)
		[47-16] : Incremented PC (forwarded)
		[79-48] : R[Rsrc1]
		[111-80] : R[Rsrc2]
		[114-112] : Rsrc1 (forwarded)
		[117-115] : Rsrc2 (forwarded)

		[118] : Rsrc1.enable (forwarded)
		[119] : Rsrc2.enable (forwarded)
		[121-120] : ALUsrc2 (00: Imm, 01:EA, 1X:Val)
		[124-122] : Rdst (forwarded)
		[128-125] : ALUcode
		[129] : Memory Read (forwarded)
		[130] : Memory Write (forwarded)
		[132-131] : Operation (00: other, 01: Swap, 10: Stack, 11: Out)
		[133] : MEM.PC (Used when updating PC from stack, RET, RTI) (forwarded)
		[134] : Rdst.WriteBack (forwarded)
		[135] : ALUOp (Used in hazards detection) (forwarded)
		[136] : MEMOp (Used in hazards detection) (forwarded)
		[137] : FR.WB (Used when updating FR from stack: RTI) (forwarded)
EX/MEM	113	[31-0] : OP1.value
		[63-32] : OP2.value
		[95-64] : Incremented PC
		[98-96] : Rdst
		[101-99] : Rsrc1 (In case of swap)
		[102] : Rsrc1.enable
		[103] : Rsrc2.enable
		[104] : Memory Read
		[105] : Memory Write
		[107-106] : Operation (00: other, 01: Swap, 10: Stack, 11: Out)
		[108] : MEM.PC (Used when updating PC from stack, RET, RTI)
		[109] : Rdst.WriteBack
		[110] : ALUOp (Used in hazards detection)
		[111] : MEMOp (Used in hazards detection)
		[112] : FR.WB (Used when updating FR from stack: RTI) (forwarded)
MEM/WB	79	[31-0] : OP1.value
		[63-32] : OP2.value
		[66-64] : Rdst
		[69-67] : Rsrc1 (In case of swap)
		[70] : Rsrc1.enable
		[71] : Rsrc2.enable

		[73-72] : Operation (00: other, 01: Swap, 10: Stack, 11: Out)
		[74] : MEM.PC (Used when updating PC from stack, RET, RTI)
		[75] : Rdst.WriteBack
		[76] : ALUOp (Used in hazards detection)
		[77] : MEMOp (Used in hazards detection)
		[78] : FR.WB (Used when updating FR from stack: RTI) (forwarded)

Pipeline Hazards:

Data Forwarding:

Stage	Condition	Solution
Fetch	Instruction is Jmp ([15-13] == 110) AND (Rdst == EX/MEM .Rdst AND EX/MEM .ALUOp)	Fetch EX/MEM .OP1
	Instruction is Jmp ([15-13] == 110) AND (Rdst == EX/MEM .Rsrc1 AND EX/MEM .Operation == Swap)	Fetch EX/MEM .OP2
	Instruction is Jmp ([15-13] == 110) AND (Rdst == ID .Rdst AND (ID .ALUOp OR ID .MEMOp))	Stall
	Instruction is Jmp ([15-13] == 110) AND (Rdst == ID .Rsrc1 AND ID .Operation == Swap)	
	Instruction is Jmp ([15-13] == 110) AND (Rdst == ID/EX .Rdst AND (ID/EX .ALUOp OR ID/EX .MEMOp))	
	Instruction is Jmp ([15-13] == 110) AND (Rdst == ID/EX .Rsrc1 AND ID/EX .Operation == Swap)	
	Instruction is Jmp ([15-13] == 110) AND (Rdst == EX/MEM .Rdst AND EX/MEM .MEMOp))	
Decode		
Execute	(ID/EX .Rsrc1 == EX/MEM .Rdst OR ID/EX .Rsrc2 == EX/MEM .Rdst) AND EX/MEM .ALUOp	Fetch EX/MEM .OP1
	(ID/EX .Rsrc1 == EX/MEM .Rsrc1 OR ID/EX .Rsrc2 == EX/MEM .Rsrc1) AND EX/MEM .Operation == Swap	Fetch EX/MEM .OP2
	(ID/EX .Rsrc1 == MEM/WB .Rdst OR ID/EX .Rsrc2 == MEM/WB .Rdst) AND MEM/WB .ALUOp	Fetch MEM/WB .OP1
	(ID/EX .Rsrc1 == MEM/WB .Rsrc1 OR ID/EX .Rsrc2 == MEM/WB .Rsrc1) AND MEM/WB .Operation == Swap	Fetch MEM/WB .OP2
	(ID/EX .Rsrc1 == MEM/WB .Rdst) AND MEM/WB .MEMOp	Fetch

		MEM/WB.OP1
	(ID/EX .Rsrc1 == EX/MEM .Rdst) AND EX/MEM .Operation == In	Fetch EX/MEM.OP1
	(ID/EX .Rsrc1 == MEM/WB .Rdst) AND MEM/WB .Operation == In	Fetch MEM/WB.OP1
	(ID/EX .Rsrc1 == EX/MEM .Rdst) AND EX/MEM .MEMop	Stall
Memory	(EX/MEM .Rsrc1 == MEM/WB .Rdst OR EX/MEM .Rsrc2 == MEM/WB .Rdst) AND MEM/WB .MEMop	Fetch MEM/WB.OP1
Write back		

Hazard Detection Unit:

1- Branch Stalling:

(2 Stalls)

POP R1	F	D	E	M	WB				
CALL R1		F	F	F	D	E	M	WB	

(1 Stall)

NOT R1	F	D	E	M	WB				
INC R1	F	D	E	M	WB				
DEC R1	F	D	E	M	WB				
SWAP R1, R2	F	D	E	M	WB				
ADD R2, R3, R1	F	D	E	M	WB				
SUB R2, R3, R1	F	D	E	M	WB				
AND R2, R3, R1	F	D	E	M	WB				
OR R2, R3, R1	F	D	E	M	WB				
LDD R1, 1651	F	D	E	M	WB				
CALL R1		F	F	D	E	M	WB		

(No Stalling)

IN R1	F	D	E	M	WB				
IADD R2, R1, F	D	E	M	WB					
9651	F	D	E	M	WB				
LDM R1, 4165	F	D	E	M	WB				
SHL R1, 5654	F	D	E	M	WB				
SHR R1, F	D	E	M	WB					

5556	F	D	E	M	WB	
CALL R1		F	D	E	M	WB

Check

1- Branch Instruction (JMP, JZ, CALL) with skip_instruction = 0 (not an EA or Imm)

2- Having same Rdst

Decode: NOT INC DEC SWAP ADD SUB AND OR POP

De(D) ALUsrc2(bit_1) = 1, RdstWB = 1, Rdst, Rsrc1, Operation = 01

EX: LDD POP

Ex(D) MEM_Read: 1, MEM_OP: 1, Rdst

2- POP Data Hazard Stalling:

POP R1	F	D	E	M	WB	
INC R1		F	D	E	E	M WB

Check

POP(in memory): MEM_Read: 1, MEM_OP: 1, Stack: 10, Rdst

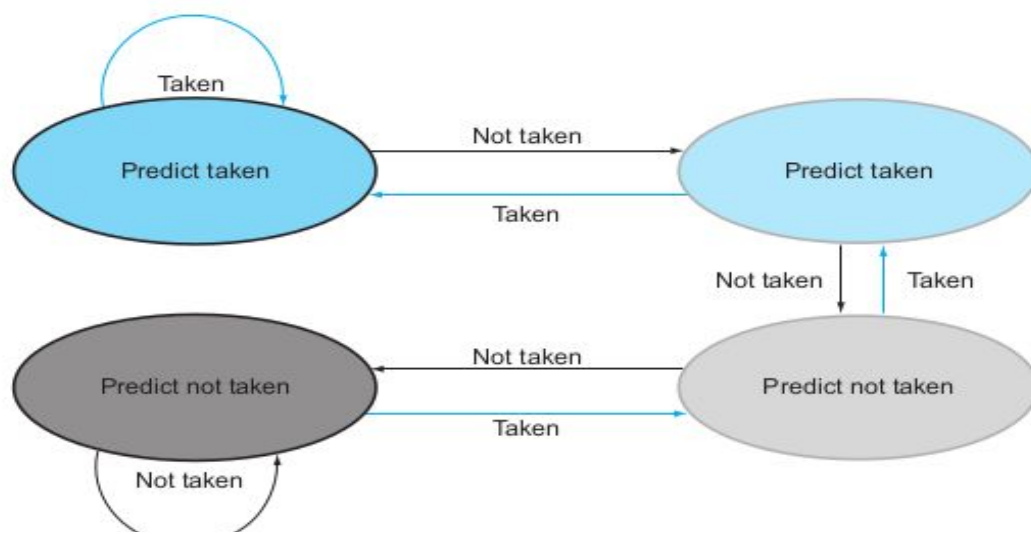
Inst(in execute): Rsrc1, Rsrc2, Rsrc1_enable, Rsrc2_enable, (to be discussed) MEM_OP = 0

Dynamic Branch Prediction:

We have a small memory that is 2-bits in width and have 2^8 places in height, and for every branch instruction, we access the memory by the lowest 8 bits of the address of the branch instruction, and store 2-bits that indicate the last state of the branch, these two bits are initialized with zeros.

The update of the 2-bits follow the following FSM:

1. 00: strongly not taken
2. 01: weakly not taken
3. 10: weakly taken
4. 11: strongly taken



Not Implemented

1. Memory cache system, we are working with two separate memories.

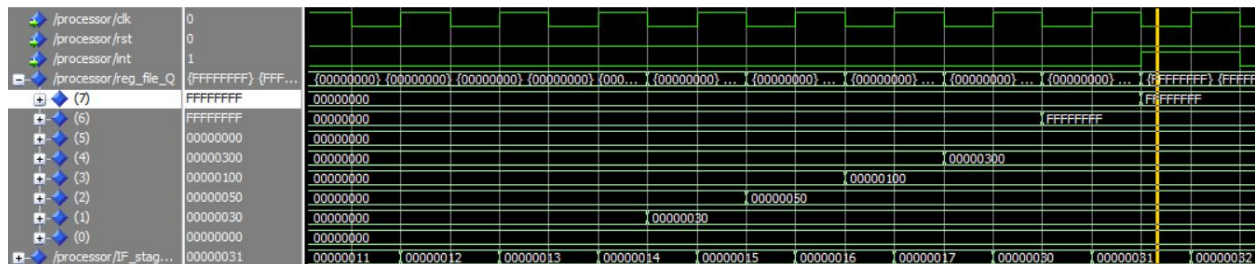
Test Cases Analysis

Branch Test:

Bare Processor: No Forwarding, Stalling, Flushing

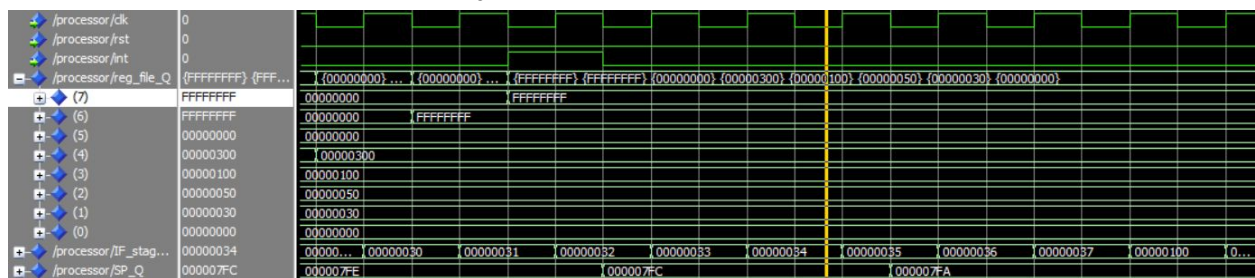
1. in R1 #R1=30
2. in R2 #R2=50
3. in R3 #R3=100
4. in R4 #R4=300
5. in R6 #R6=FFFFFFFF
6. in R7 #R7=FFFFFFFF

No problems here as there are no possible hazards. All registers are updated successfully.



7. Push R4 #sp=7FC, M[7FE, 7FF]=300
8. JMP R1
9. INC R7 # this statement shouldn't be executed,

No hazards here as well as there are no data hazards on R1 before Push instruction or control hazards from JMP instruction. INC R7 statement does not enter the pipeline as the previous instruction is a direct (unconditional) jump.



10. ..ORG 30
11. AND R1,R5,R5 #R5=0 , Z = 1
12. #try interrupt here
13. JZ R2 #Jump taken, Z = 0
14. INC R7 #this statement shouldn't be executed

No hazards in AND instruction. JZ is not added to the pipeline as the interrupt signal has been raised before it enters the pipeline. However, there is a data hazard over the Stack Pointer

when pushing both PC and flags during interrupt handling. Since the flags are pushed second, they will overwrite the PC value stored. This will not have a direct effect on the interrupt part, however, this will cause RTI statements to return to a false address.

(2042)	0000	0000																		
(2043)	0000	0000																		
(2044)	0000	0000																		
(2045)	0001	0000								0031		0001								
(2046)	0000	0000																		
(2047)	0300	0000			0300															

15. .ORG 100
16. ADD R0,R0,R0 #N=0,Z=1,C=0
17. Out R6
18. RTI

No hazards in interrupt but as stated previously, RTI statements will cause PC to return to a false address.

/processor/IF_stage/curr_address	00000001	000...	00000102	00000103	00000104	00000105	00000106	00000107	00000001	00000002
----------------------------------	----------	--------	----------	----------	----------	----------	----------	----------	----------	----------

This can only be fixed by adding the Forwarding unit as the interrupt instructions (Pushing PC, flags and retrieving new PC from [2, 3]) are inserted into the pipeline successively. In the meantime, PC returned to value 0001 which contains the value of the reset address but will be treated as a new instruction and will produce non-meaningful results.

Add Forwarding Unit

All the previous instructions are working correctly including interrupt and RTI instructions. This time, Stack Pointer was forwarded after its value was updated in the pipeline but had not reached the Write Back stage, yet. This causes the PC to return to JZ and resume normal operations.

/processor/IF_stage/curr_address	00000031	00000101	00000102	00000103	00000104	00000105	00000106	00000107	00000031	00000032	00000050
----------------------------------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

1. .ORG 30
2. AND R1,R5,R5 #R5=0 , Z = 1
3. #try interrupt here
4. JZ R2 #Jump taken, Z = 0
5. INC R7 #this statement shouldn't be executed

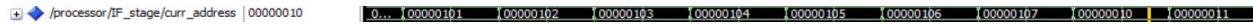
Branch prediction predicts not taken while conditional jumps (JZ) are finalised during decode stage since the correct value of zero flag will be available by then. At the same time, INC instruction entered the pipeline and is in the fetch stage. As this version of the processor has no flushing, then this instruction remains in the pipeline and causes an unwanted change in the value of R7. This can be solved by adding a single NOP instruction after JZ.

(7)	00000000	FFFFFFFF								00000000
(6)	FFFFFFFF									
(5)	00000000	00000000								
(4)	00000300	00000300								
(3)	00000100	00000100								
(2)	00000050	00000050								
(1)	00000030	00000030								
(0)	00000000	00000000								
/processor/IF_stage/curr_address	00000101	00000106	00000107	00000031	00000032	00000050	00000051	00000100	00000101	00000102
/processor/SP_Q	000007FC	000...	000007FA	000007FC						
/processor/R7_Q	6	1				0	5	4	6	1

6. .ORG 50

7. JZ R3 #Jump Not taken

In addition to the unwanted change in the value of R7, incrementing R7 caused its value to be 0, and subsequently, the zero flag to be 1. This causes the JZ R3 instruction to be taken, jumping to address 100 which is the same as interrupt. This will cause the same issue that happened in the previous version of the processor, to return a false value as PC.



8. #check destination forwarding

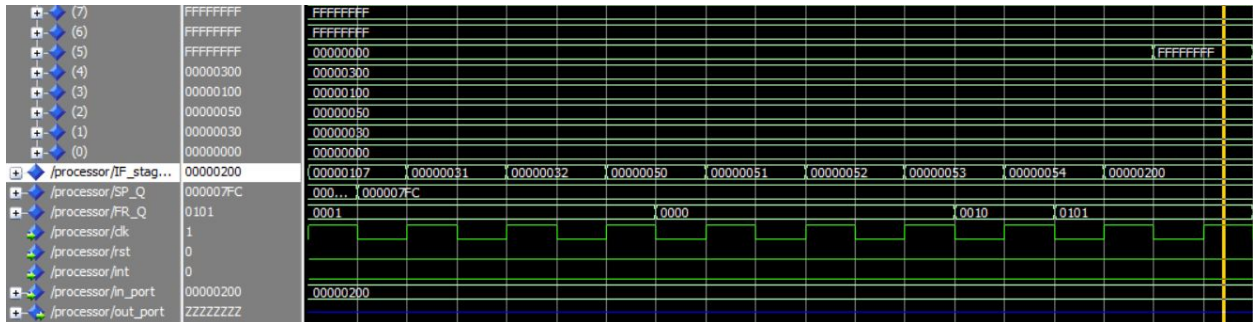
9. NOT R5 #R5=FFFFFFF, Z= 0, C--> not change, N=1

10. INC R5 #R5=0, Z=1, C=1, N=0

11. in R6 #R6=200, flag no change

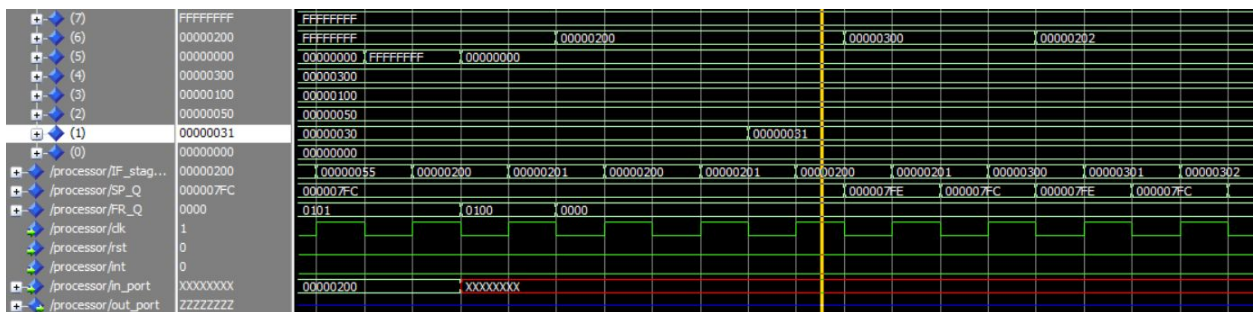
12. JZ R6 #jump taken, Z = 0

After adding one NOP, everything is working fine. INC R7 never entered the pipeline nor changed the zero flag to 1. Thus, JZ R3 was never taken either. After that, only data hazards arise between NOT R5, INC R5. This doesn't generate any problems in this version where forwarding is allowed. The same applies to IN R6 and JZ R6.



13. INC R1 #this statement shouldn't be executed

Like the last time, INC R1 enters the pipeline before the actual jumping occurs, changing the value of R1. It can be solved by also adding a single NOP instruction.



14. .ORG 200

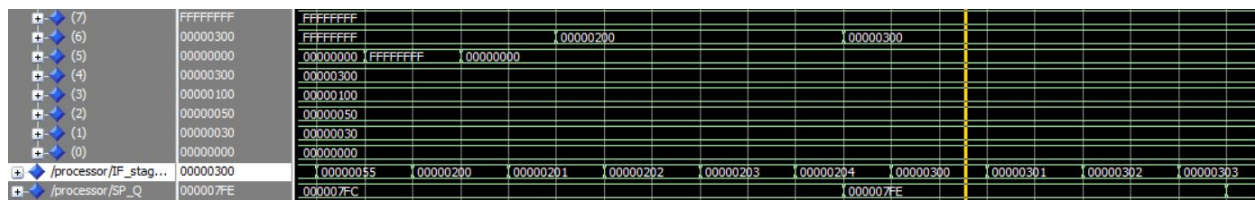
15. POP R6 #R6=300, SP=7FE

16. Call R6 #SP=7FC, M[7FF]=half next PC, M[7FE]=other half next PC

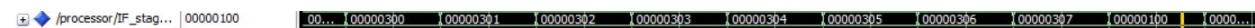
17. #try interrupt here

There is a data hazard between the POP instruction and the Call instruction. This cannot be solved by forwarding the data as the data has not been retrieved yet. It will be retrieved at the memory stage (Load use case). This requires stalling the Call instruction until the data is

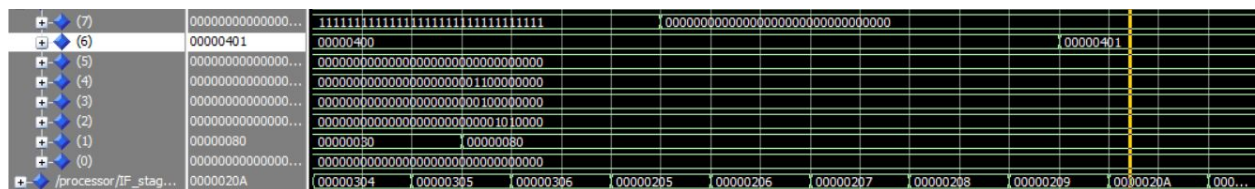
retrieved. This causes the PC to get stuck in a loop jumping between POP and Call instructions until the new value is available and adding multiple POP R6 instructions that shouldn't be added. This can be solved temporarily by adding 3 NOP operations until the data is available.



After adding 3 NOP operations, the PC goes straight to address 300, the start of the call, then signal interrupt is raised and it goes to 100, the start of the interrupt.



After going to interrupt and returning, the PC resumes in the Call part and returns correctly as well. In the end, the value of R6 is 401 as it should.

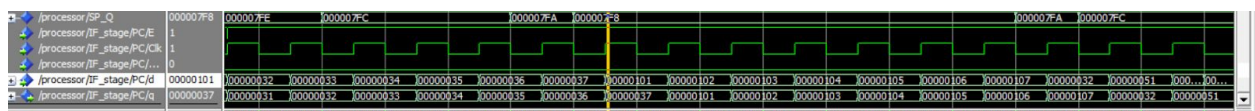


Add Stalling

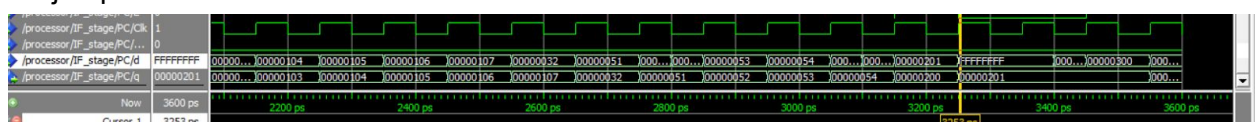
After removing some of the extra NOP's that were corresponding to Stall signals, the value is still (correctly) 401 in R6. Only the NOP's that correspond to flushing operations are left.

Adding Flushing

At .ORG 30 The interrupt takes place after ADD then it goes to .ORG 100 executing interrupt service routine, popping the flags and returning to the jump instruction it will predict not taken but quickly it will flush the wrong new PC and take the right decision to jump to .ORG 50 and reset the Z flag



After that the branch prediction will predict not taken so it will not take branch in .ORG 30 which was a right decision so no flushing occurs, and the next instructions will continue till JZ R6, it will predict not taken but the write decision s to be taken so the wrong PC will be flushed. And the jump occurs to .ORG 200.




INC R0	F	D	E	M	W
AND R0,R2,R5 #when R0 < R2(8) answer will be zero	F	D	E	M	W

+	2	00000000000000000000000000000000 1000	00000000000000000000000000000000 1000
+	1	00000000000000000000000000000000 1010000	00000000000000000000000000000000 1010000
+	0	00000000000000000000000000000000 1001	00000000000000000000000000000000 1001

AND R0,R2,R5 #when R0 < R2(8) answer will be zero	F	D	E	M	W	
JZ R3 #jump if R0 < R2 to 60 , R3=60	F	F	D	E	M	W
INC R4			F	D	E	M W

[illegible][illegible]

 /processor/out_port 00000000000000000000000011000000

	(5)	(4)	(3)	(2)	(1)	(0)
+	00000000000000000000000000000000	11111111111111111111111111111111	00000000000000000000000000000000			
+	00000000000000000000000000000000	00000000000000000000000000000000	10000000000000000000000000000000			
+	00000000000000000000000000000000	00000000000000000000000000000000	100000			
+	00000000000000000000000000000000	00000000000000000000000000000000	10 10			
+	00000000000000000000000000000000	00000000000000000000000000000000	10 1000			
+	00000000000000000000000000000000	00000000000000000000000000000000	10 10			

The loop will execute the correct number of times and now when R5 = 0, R0 = R2.

Error #4, the data hazard here is resolved and the loop runs for the correct number of iterations and R0 = R2 at the end.

+	(5)	00000000000000000000000000000000 1000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000
+	(4)	00000000000000000000000000000000 1111000000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000
+	(3)	00000000000000000000000000000000 11000000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000
+	(2)	00000000000000000000000000000000 100000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000
+	(1)	00000000000000000000000000000000 10 10000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000
+	(0)	00000000000000000000000000000000 1000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000

(4)	00000000000000000000000000000000111	00000000000000000000000000000000110	0000000000...000000000000000000000000000000001110
-----	-------------------------------------	-------------------------------------	---

[illegible]

+	/processor/out_port	000000000000000000000000111100000001	000000000000...	000000000000...
---	---------------------	--------------------------------------	-----------------	-----------------

+ /processor/IF_stage/PC/q	101	97	98	99	100	101	96	97
+ /processor/IF_stage/force_pc	0							

+ /processor/IF_stage/PC/q	96	96	97	98	99	100	96	101
/processor/IF_stage/force_pc	1							

[illegible]

Memory Test:

Basic Processor:

1. .ORG 10:

LDM R1, F5 F D E M W

PUSH R1	F	D	E	M	W.
---------	---	---	---	---	----

The LDM is two operand-instruction so the pipe should stall and the next instruction will be Push.

When the PUSH was at decode stage the LDM was still at memory and decoded value was a past wrong value not that from LDM so it gets zero not F5 and pushes it in the memory stack. A data hazard here exists, This can be handled by adding a NOP.

[illegible]

2. PUSH R1 F D E M W

PUSH R2 F D E M W

POP R1	F	D	E	M	W
--------	---	---	---	---	---

POP R2	F	D	E	M	W
--------	---	---	---	---	---

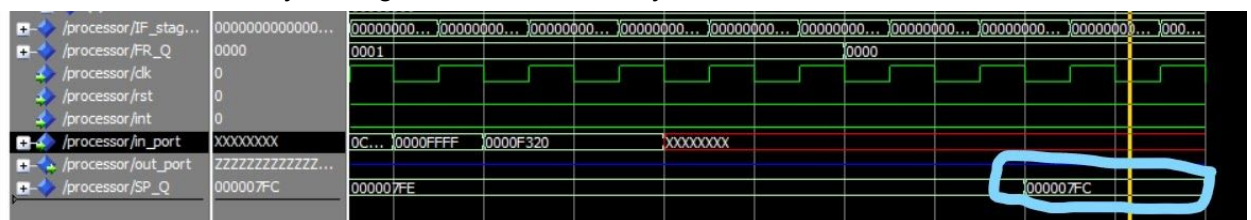
SP = 7FE at first

The second PUSH reads the wrong value of SP as it will be written in WB stage so it reads it with old value SP = 7FE. So both PUSH instructions read the same value of SP and will write in the same place in the memory.

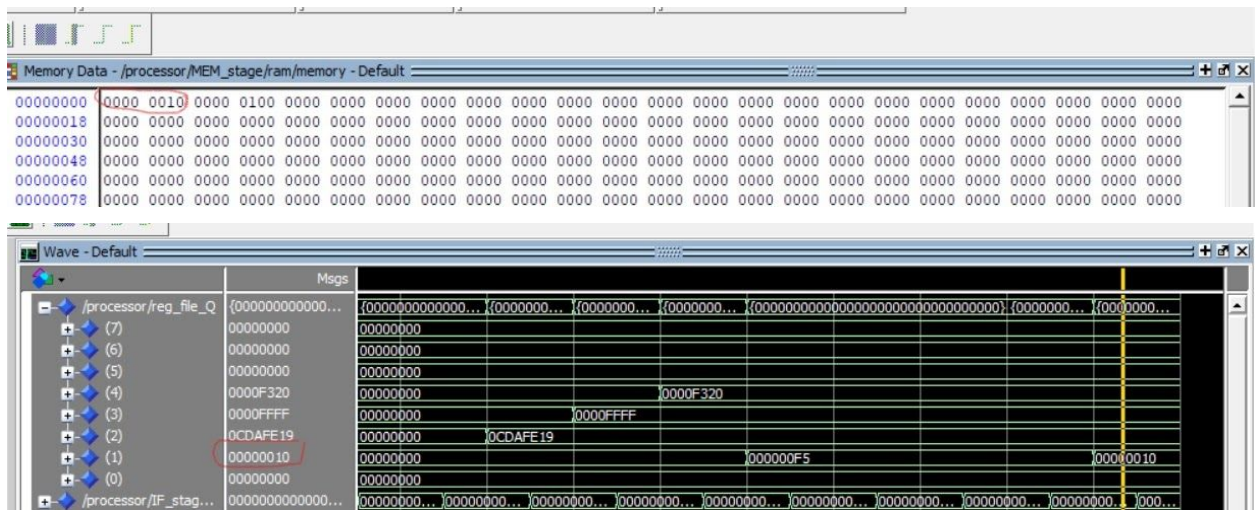
After two pushes the place in memory 7FE holds R2,M[7FE]=0CDAFE19 and SP=7FC.

The effect of SP decrement will be done once due to the data hazard RAW.

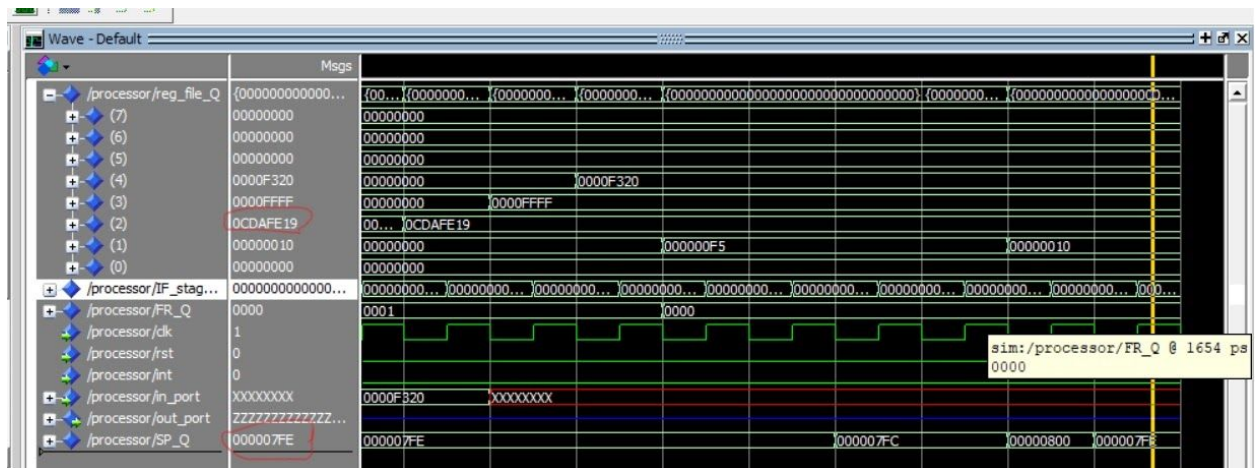
This can be handled by adding two NOP after every instruction.

[illegible]

So will do the first POP it will read SP=7FE before any of the above instructions modify it so it will overflow(SP = 0) and take the value located at M[0:1] which is 2 and place it and put it in R1 = 2.



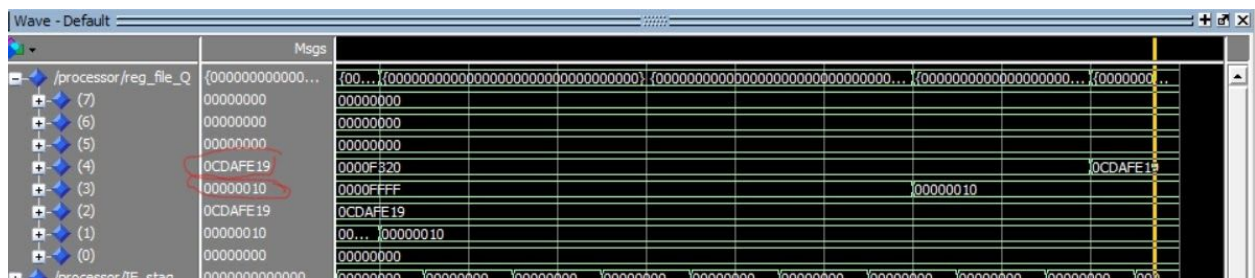
The second POP will be in decode stage while first POP in writeback so it will take value of stack pointer written by first push SP = 7FC and will change it to SP = 7FE and put M[7FE:7FF] in R2, R2 = 0CDAFE19 which was previously at R2.



3. The next store R2, and store R1, store values of R2 = 0CDAFE19, R1 = 2 in places M[200, 201], M[202, 203] respectively.

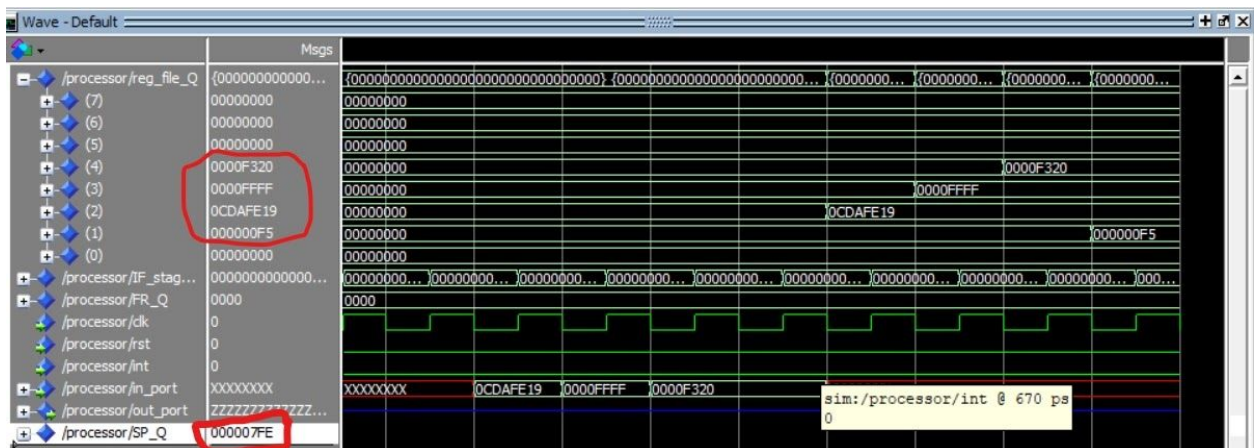


4. The next load s will load value located at M[202, 203], M[200, 201] to R3, R4 respectively So R3 = M[202, 203] = 2, R4 = M[200, 201] = 0CDAFE19.

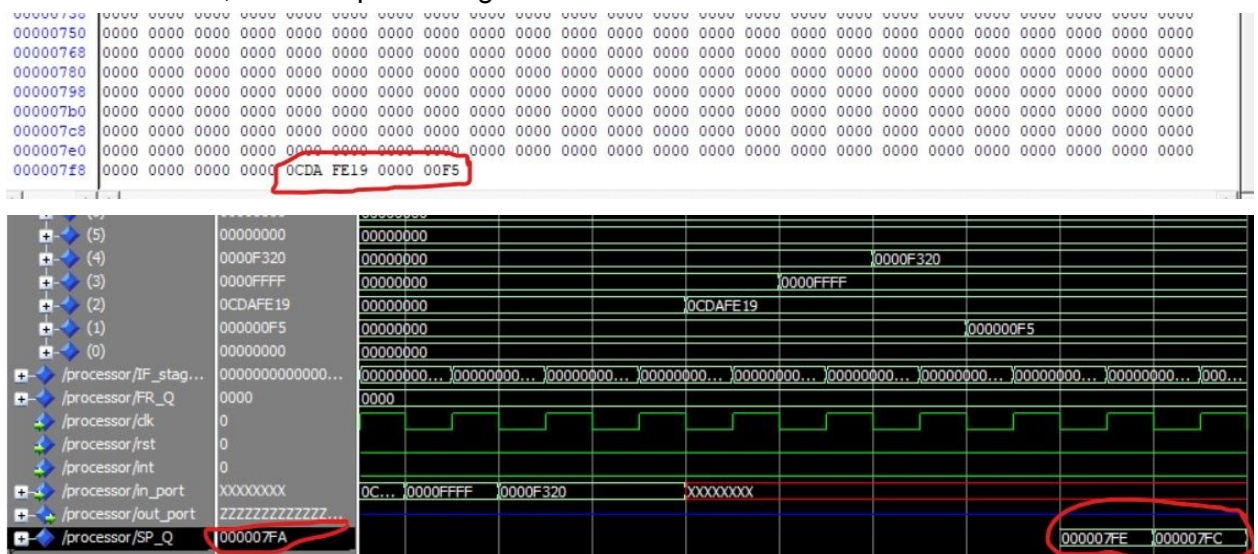


Adding Forwarding Unit:

- All previous data hazards are handled except for POP then STD as the result of POP will be ready after memory so it has to stall.
- After loading R1, R2, R3, R4 from in port and LDM instruction.



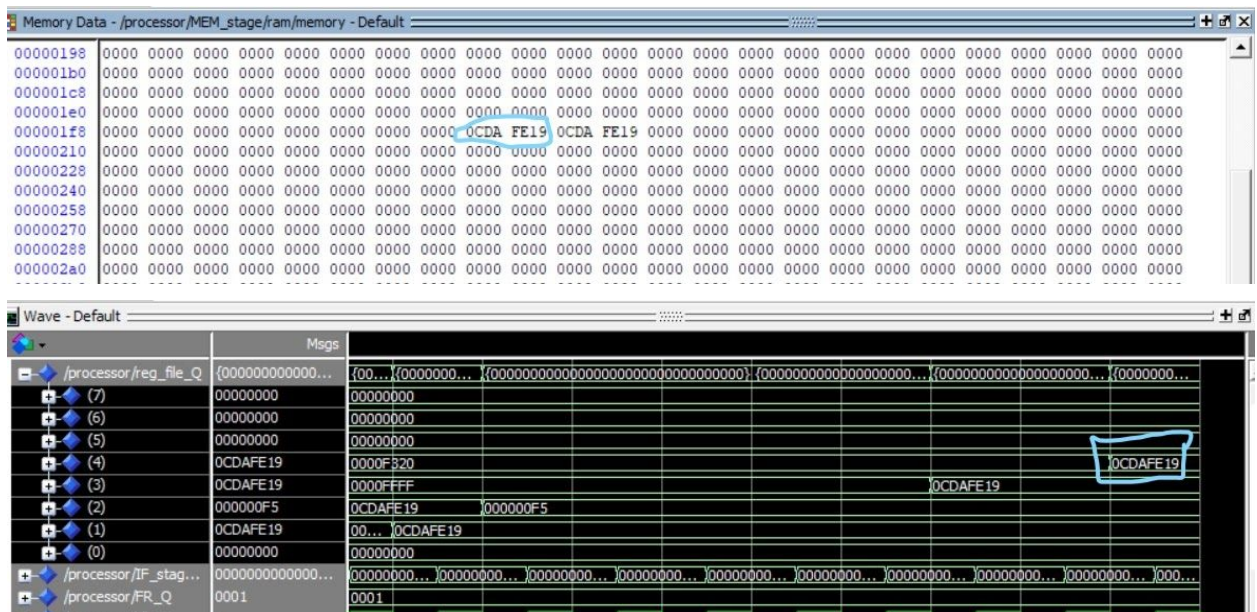
- The values of R1, R2 were pushed right in the stack and the SP decremented twice.



- The register values of R1, R3, are switched by the two pop instructions and the SP incremented twice.



- A hazard takes place and the old value of R1 is written in the memory M[200:201] not that from the POP and so the writing on R4 a wrong value



Adding Hazard Detection Unit:

- All Previous hazards were overcome by adding the hazard detection unit that stalls in case of memory read instruction followed by an instruction that needs the result of memory read then it is stalled in decode till the result of memory is ready.
- R1, R2, R3, R4 are loaded from in port and LDM instruction.
 - The values of R1, R2 were pushed right in the stack and the SP decremented twice.
 - The register values of R1, R3, are switched by the two pop instructions and the SP incremented twice.

/processor/rst	0						
/processor/in_port	XXXXXXXX	XXXXXXXX	00000005	00000010	XXXXXXXX		
/processor/out_port	ZZZZZZZZ						
/processor/reg_fil...	{000000000000...	{00000...	{000000000000000000...	{00000...	{00000...	{00000...	{00000...
(7)	00000000	00000000					
(6)	00000000	00000000					
(5)	00000000	00000000					
(4)	00000000	00000000					
(3)	00000000	00000000					
(2)	0000000F	00000000			00000010	FFFFFFFF	
(1)	00000006	00000000	FFFFFFFF	00000001	00000005		
(0)	00000000	00000000					
/processor/FR_Q	0000	0010	0000		0010	0000	

Instr.4 and 5 are executed normally and R1, R2 are updated with 5, 10 respectively
 There is a data hazard in instr.6 on R2 and 2 NOP should be inserted between instr.5 and instr.6 R2 is still seen as 0, so it's updated to FFFFFFFF and N flag is 1

- ```

7) inc R1 #R1= 6, C --> 0, N -->0, Z-->0
8) Dec R2 #R2= FFFFFFFE,C-->1 , N-->1, Z-->0
9) out R1
10) out R2

```

Logic Analyzer tool showing the processor's internal state. The left pane displays the register file (reg\_fil...) with values for registers (7) through (0). The right pane displays the processor's internal state, including the program counter (PC) and the instruction register (IR). The PC is 00000010, and the IR contains the instruction 00000005 FFFFFFFF. The instruction register is highlighted in red.

Instr.7 is executed correctly and R1 = 5, FR = 000

Instr.8 has data hazard on R2, and NOP should be inserted before it, R2 is still seen as 10, so it's decremented to 0F and FR is still 0000.

Instr.9 has data hazard on R1, and NOP should be inserted before it, R1 is still seen as 5

Instr.10 has data hazard on R2, and NOP should be inserted before it, R2 is still seen as FFFFFFFF

With Forwarding unit:

- 1) NOT R1      #R1 =FFFFFFF , C--> no change, N --> 1, Z --> 0
- 2) NOP        #No change
- 3) inc R1     #R1 =00000000 , C --> 1 , N --> 0 , Z --> 1

The screenshot displays the Logic Analyzer tool interface. The left pane shows the processor state, including the in\_port, out\_port, reg\_file\_Q, and FR\_Q registers. The right pane shows the memory state, with addresses 17 through 23. Red annotations highlight specific data values and their corresponding memory addresses.

| Register/Address              | Value                                 |
|-------------------------------|---------------------------------------|
| /processor/in_port            | XXXXXXX                               |
| /processor/out_port           | ZZZZZZZ                               |
| /processor/reg_file_Q         | {00000000000000000000000000000000...} |
| (7)                           | 00000000                              |
| (6)                           | 00000000                              |
| (5)                           | 00000000                              |
| (4)                           | 00000000                              |
| (3)                           | 00000000                              |
| (2)                           | FFFFFFEF                              |
| (1)                           | 00000005                              |
| (0)                           | 00000000                              |
| /processor/FR_Q               | 0010                                  |
| /processor/IF_stage/curr_a... | 25                                    |

| Address | Value    |
|---------|----------|
| 17      | XXXXXXX  |
| 18      | 00000005 |
| 19      | 00000010 |
| 20      | XXXXXXX  |
| 21      | 00000000 |
| 22      | 00000000 |
| 23      | 00000000 |

Red annotations highlight the following data values and their corresponding memory addresses:

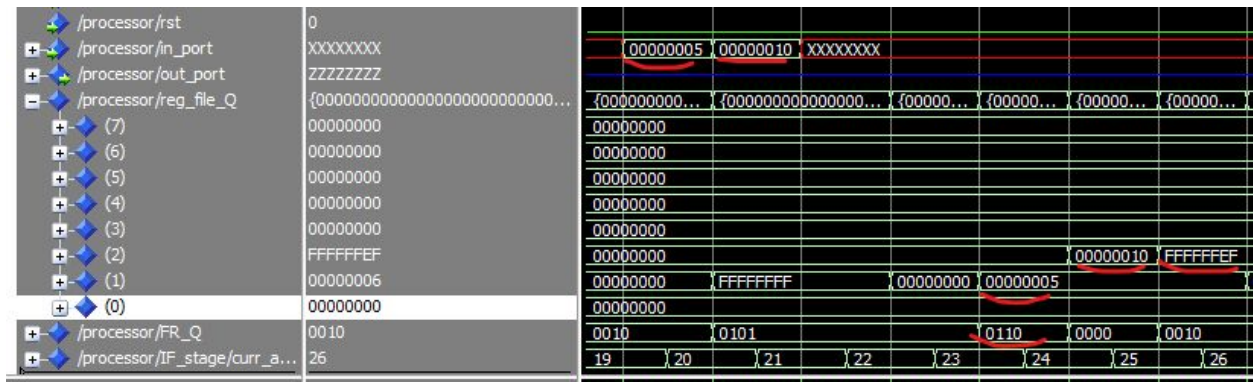
- FFFFFFFFFF at address 21
- 00000000 at address 22
- 0010 at address 18
- 0101 at address 21

Instr.1 is executed normally and R1 = FFFFFFFF, FR = 010

Instr.3 has data hazard on R1, but R1 is forwarded from memory stage, so it's updated normally and = 0, FR = 101.

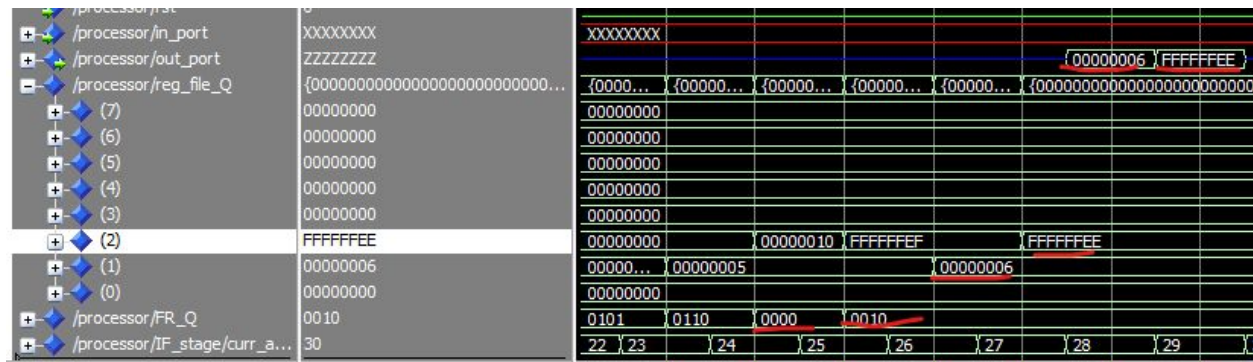


- 4) in R1            #R1= 5,add 5 on the in port,flags no change
- 5) in R2            #R2= 10,add 10 on the in port, flags no change
- 6) NOT R2          #R2= FFFFFFFEF, C--> no change, N -->1,Z-->0



Instr. 4 and 5 are executed normally and R1, R2 are updated to 5, 10 respectively  
 Instr.6 has data hazard on R2 but it's forwarded from execute stage and R2 is updated to FFFFFFFEF and FR = 110.

- 7) inc R1            #R1= 6, C --> 0, N -->0, Z-->0
- 8) Dec R2            #R2= FFFFFFFEE,C-->1 , N-->1, Z-->0
- 9) out R1
- 10) out R2



Instr.7 is executed normally, and R1 = 6, FR = 000  
 Instr.8 has data hazard on R2 because it was modified on instr.6, but it's forwarded from memory stage, and R2 is updated correctly to FFFFFFFEE, and FR = 010 (no carry, negative, not zero).  
 Instr.9 has data hazard on R1 because it's updated in instr.7, but it's forwarded from memory stage to execute stage and the out port has the correct value 6  
 Instr.10 has data hazard on R2 because it's updated in instr.8, but it's forwarded from memory stage to execute stage and the out port has the correct value FFFFFFFEE

## Two Operand Test:

### Basic Processor:

- 1) in R1 #add 5 in R1
- 2) in R2 #add 19 in R2
- 3) in R3 #FFFD
- 4) in R4 #F320

| Register | Value    |
|----------|----------|
| 0        | 00000000 |
| 1        | 00000005 |
| 2        | 00000019 |
| 3        | 0000FFFD |
| 4        | 0000F320 |
| 5        | 00000000 |
| 6        | 00000000 |
| 7        | 00000000 |

Instr. 1, 2, 3, 4 are executed correctly and R1, R2, R3, R4 are updated correctly

- 5) IADD R3,R5 #R5 = FFFF , flags no change
- 6) 2 #immediate value
- 7) ADD R1,R4,R4 #R4= F325 , C-->0, N-->0, Z-->0
- 8) SUB R5,R4,R6 #R6= 0CDA , C-->1, N-->0,Z-->0

| Register | Value    |
|----------|----------|
| 0        | 00000000 |
| 1        | 00000005 |
| 2        | 00000019 |
| 3        | 0000FFFD |
| 4        | 0000F320 |
| 5        | 00000002 |
| 6        | 00000000 |
| 7        | 00000000 |

Instr.5 has data hazard on R3 because it's updated in instr.3, NOP should be inserted before it R3 is still seen as 0 so  $R5 = 0 + 2 = 2$  and FR updated in execute stage with 000.

Instr.7 executed correctly and  $R4 = R1 + R4 = F325$ , FR = 000

Instr. 8 has data hazard on R4, 2 NOPs should be inserted before it. R4 is still seen as F320, so  $R6 = R5 - R4 = 2 - F320 = FFFF0CE2$ , and FR = 110 (negative and carry flag = 1)

- 9) AND R7,R6,R6 #R6= 00000000 , C-->no change, N-->0, Z-->1  
 10) OR R2,R1,R1 #R1=1D , C--> no change, N-->0, Z--> 0  
 11) SHL R2 #R2=64 , C--> 0, N -->0 , Z -->0  
 12) 2 #immediate value

|                               |                                       |                       |               |               |               |               |                                       |
|-------------------------------|---------------------------------------|-----------------------|---------------|---------------|---------------|---------------|---------------------------------------|
| /processor/reg_file_Q         | {00000000000000000000000000000000...} | {0000000000000000...} | {00000000...} | {00000000...} | {00000000...} | {00000000...} | {00000000000000000000000000000000...} |
| (7)                           | 00000000                              | 00000000              |               |               |               |               |                                       |
| (6)                           | 00000000                              |                       | FFFF0CE2      | 00000000      |               |               |                                       |
| (5)                           | 00000002                              |                       |               |               |               |               |                                       |
| (4)                           | 0000F320                              |                       | 0000F325      |               |               |               |                                       |
| (3)                           | 0000FFFD                              |                       |               |               |               |               |                                       |
| (2)                           | 00000019                              |                       |               |               |               |               | 00000064                              |
| (1)                           | 00000005                              |                       |               |               |               |               | 0000001D                              |
| (0)                           | 00000000                              |                       |               |               |               |               |                                       |
| /processor/FR_Q               | 0000                                  |                       |               |               |               |               |                                       |
| /processor/IF_stage/curr_a... | 24                                    | 0110                  | 0001          | 0000          |               |               |                                       |
|                               |                                       | 24                    | 25            | 26            | 27            | 28            | 29                                    |

Instr.9 has data hazard on R6 because it's updated in instr.8, 2 NOPs should be inserted before it, R6 is still seen as 0, but this the result is not affected and R7 = 0, FR = 001

Instr.10 is executed correctly and R1 = 1D, FR = 000

Instr. 11 is executed correctly and R2 = 64 and FR = 000

- 13) SHR R2 #R2=0C , C -->1, N-->0 , Z-->0  
 14) 3 #immediate value  
 15) SWAP R2,R5 #R5=0C ,R2=FFFF ,no change for flags  
 16) ADD R5,R2,R2 #R2= 1000B (C,N,Z= 0)

|                               |                                       |          |                               |                               |               |                                       |
|-------------------------------|---------------------------------------|----------|-------------------------------|-------------------------------|---------------|---------------------------------------|
| /processor/out_port           | 00000000                              | {0...}   | {000000000000000000000000...} | {000000000000000000000000...} | {00000000...} | {00000000000000000000000000000000...} |
| /processor/reg_file_Q         | {00000000000000000000000000000000...} | 00000000 | 00000000                      | 00000000                      | 00000000      | 00000000                              |
| (7)                           | 00000000                              | 00000000 |                               |                               |               |                                       |
| (6)                           | 00000000                              |          |                               |                               |               |                                       |
| (5)                           | 00000002                              |          |                               |                               |               | 00000064                              |
| (4)                           | 0000F320                              |          |                               |                               |               | 0000F325                              |
| (3)                           | 0000FFFD                              |          |                               |                               |               | 0000FFFD                              |
| (2)                           | 00000019                              |          |                               |                               |               | 00...                                 |
| (1)                           | 00000005                              |          | 00000064                      | 00000003                      | 00000002      | 00000005                              |
| (0)                           | 00000000                              |          | 0000001D                      |                               |               |                                       |
| /processor/FR_Q               | 0000                                  | 0000     |                               |                               |               |                                       |
| /processor/IF_stage/curr_a... | 24                                    | 30       | 31                            | 32                            | 33            | 34                                    |

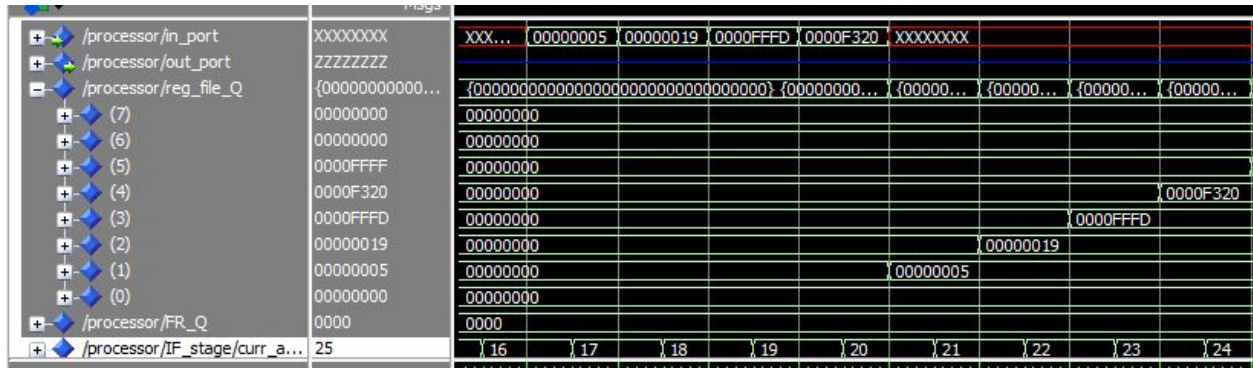
Instr. 13 has data hazard on R2 because it's updated in instr.11, a NOP should be inserted before it. R2 is still seen as 19 and is updated to 3 and FR = 000

Instr.15 has data hazard on R2 because it's updated in instr.13, a NOP should be inserted before it. R2 is still seen as 64, so R5 = 64, and R2 = 2 and FR = 000

Instr. 16 has data hazard on R5 and R2 because they are updated in instr.15, 2 NOP's should be inserted before it. R2 is still seen as 3 and R5 is still seen as 2, so R2 is updated to 5 and FR = 000.

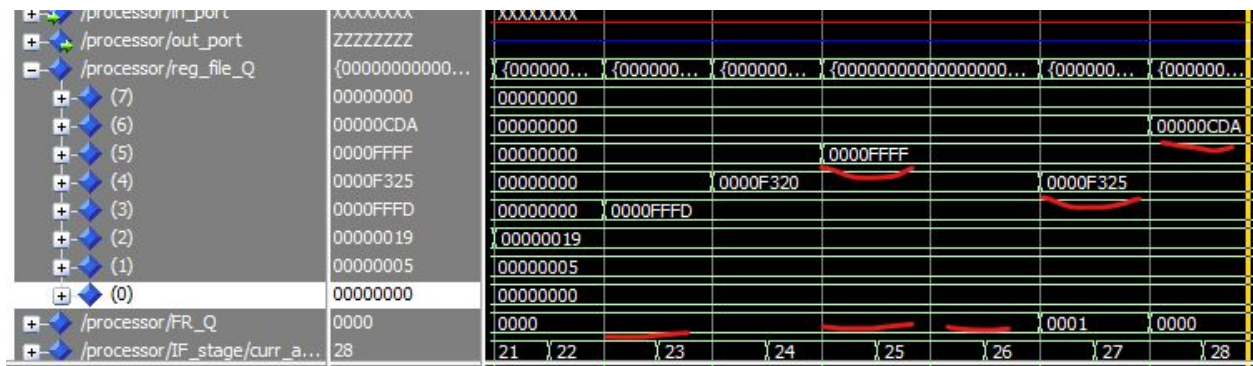
With Forwarding unit:

- 1) in R1 #add 5 in R1
- 2) in R2 #add 19 in R2
- 3) in R3 #FFFD
- 4) in R4 #F320



The first 4 instructions are executed normally, and registers are updated correctly

- 5) IADD R3,R5 #R5 = FFFF , flags no change
- 6) 2 #immediate value
- 7) ADD R1,R4,R3 #R4= F325 , C-->0, N-->0, Z-->0
- 8) SUB R5,R4,R6 #R6= 0CDA , C-->1, N-->0, Z-->0



Instr.5 has data hazard on R3 because it's updated in instr.3, but R3 is forwarded from memory stage to execute stage and R5 is updated correctly R5 = FFFF, FR = 000

Instr.7 is executed correctly R4 = F325, FR = 000

Instr.8 has data hazard on R4 because it's updated in instruction 7, but R4 is forwarded from execute stage and R6 is updated correctly to 0CDA and FR = 000



- 12) 2                      #immediate value

| Component                     | Value            |
|-------------------------------|------------------|
| /processor/in_port            | ZZZZZZZZ         |
| /processor/out_port           | {000000000000... |
| /processor/reg_file_Q (7)     | 00000000         |
| /processor/reg_file_Q (6)     | 00000000         |
| /processor/reg_file_Q (5)     | 0000FFFF         |
| /processor/reg_file_Q (4)     | 0000F325         |
| /processor/reg_file_Q (3)     | 0000FFFD         |
| /processor/reg_file_Q (2)     | 00000064         |
| /processor/reg_file_Q (1)     | 0000001D         |
| /processor/reg_file_Q (0)     | 00000000         |
| /processor/FR_Q               | 0100             |
| /processor/IF_stage/curr_a... | 31               |

stage and  $R6 = 0$ ,  $FR = 001$

Instr.10 is executed correctly and R1 = 1D, FR = 000

Instr. 11 is executed correctly and R2 = 64 and FR = 000

- 16) ADD R5,R2,R2      #R2= 1000B (C,N,Z= 0)

The screenshot shows the Logic Analyzer interface. On the left, the signal tree lists the following signals:

- /processor/in\_port
- + /processor/out\_port
- /processor/reg\_file\_Q
- + (7)
- + (6)
- + (5)
- + (4)
- + (3)
- + (2)
- + (1)
- + (0)
- + /processor/FR\_Q
- + /processor/IF\_stage/curr\_a...

The waveform on the right displays the digital signals over 36 clock cycles. The signals shown are:

- XXXXXX (top signal, mostly high)
- ZZZZZZZZ (second signal, mostly high)
- {000000000000...} (third signal, mostly low)
- 00000000 (fourth signal, mostly low)
- 00000000 (fifth signal, mostly low)
- 0000000C (sixth signal, mostly low)
- 0000FFFF (seventh signal, mostly low)
- 0000F325 (eighth signal, mostly low)
- 0000FFFD (ninth signal, mostly low)
- 00000019 (tenth signal, mostly low)
- 00000064 (eleventh signal, mostly low)
- 0000000C (twelfth signal, mostly low)
- 0000FFFF (thirteenth signal, mostly low)
- 0001000B (fourteenth signal, mostly low)
- 0000001D (fifteenth signal, mostly low)
- 00000005 (sixteenth signal, mostly low)
- 0000001D (seventeenth signal, mostly low)
- 00000000 (eighteenth signal, mostly low)
- 0000 (nineteenth signal, mostly low)
- 0100 (twentieth signal, mostly low)
- 0000 (twenty-first signal, mostly low)

The waveform also shows clock cycles 28 through 36 at the bottom.

memory stage and R2 = 0C, FR = 100

memory stage and R5 = 0C, R2 = FFFF, FR = 000

from execute stage and R2 = 1000B, FR = 000