

PIPELINE VERIFICATION

We have to make sure that our Pipeline implementation correctly resolves all the hazards. There are 3 types of hazards that occur

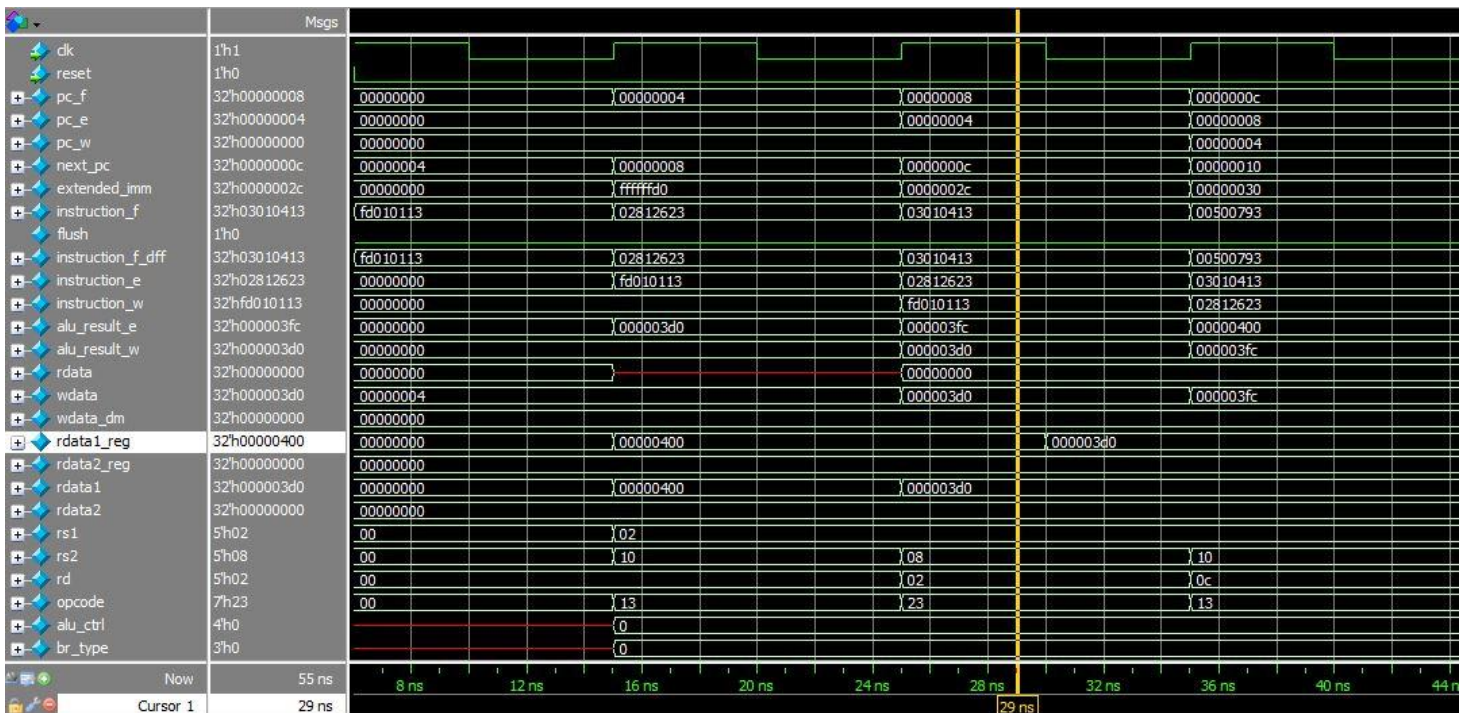
- Data Hazard
- Load Hazard
- Control Hazard

Data Hazard

Data Hazard occurs when one instruction uses the destination register of the previous instruction as operand. The instructions demonstrating Data Hazard in this example are

```
0:    fd010113          addi    sp,sp,-48
4:    02812623          sw      s0,44(sp)
```

Simulation



Load Hazard

```
34: 40f707b3      sub    a5,a4,a5
38: fcf42a23      sw     a5,-44(s0)
```

		Msgs																
clk	1h1																	
reset	1h0																	
pc_f	32h00000038	0000002c	00000030	00000034	00000038	0000003c												
pc_e	32h00000034	00000028	0000002c	00000030	00000034	00000038												
pc_w	32h00000030	00000024	00000028	0000002c	00000030	00000034												
next_pc	32h0000003c	00000030	00000034	00000038	0000003c	00000040												
extended_imm	32h0000040f	ffffffd8	ffffffe0	0000040f	ffffffd4													
instruction_f	32hfc42a23	fd842703	fe042783	40f707b3	fcf42a23	fe042623												
flush	1h0																	
instruction_f_dff	32hfc42a23	fd842703	fe042783	40f707b3	fcf42a23	fe042623												
instruction_e	32h40f707b3	fcf42c23	fd842703	fe042783	40f707b3	fcf42a23												
instruction_w	32hfe042783	00f707b3	fcf42c23	fd842703	fe042783	40f707b3												
alu_result_e	32h0000000a	000003d8	000003e0	0000000a	000003d4													
alu_result_w	32h000003e0	0000000f	000003d8	000003e0	0000000a													
rdata	32h00000005	00000000	0000000f	00000005	00000000													
wdata	32h00000005	0000000f	000003d8	0000000f	0000000a													
wdata_dm	32h00000000	0000000a	0000000f	00000000	00000005													
rdata1_reg	32h0000000f	00000400			0000000f	00000400												
rdata2_reg	32h0000000f	0000000f	00000000		0000000f	00000005	00000005		0000000a									
rdata1	32h0000000f	00000400			0000000f				00000400									
rdata2	32h00000005	0000000f	00000000		00000005				0000000a									
rs1	5h0e	08			0e				08									
rs2	5h0f	0f	18	00	0f													
rd	5h0f	0f	18	0e	0f													
opcode	7h33	23	03		33				23									
alu_ctrl	4h1	0			1				0									
br_type	3h0	0																
Now		355 ns																
Cursor 1		149 ns	124 ns	128 ns	132 ns	136 ns	140 ns	144 ns	148 ns	149 ns	152 ns	156 ns	160 ns					

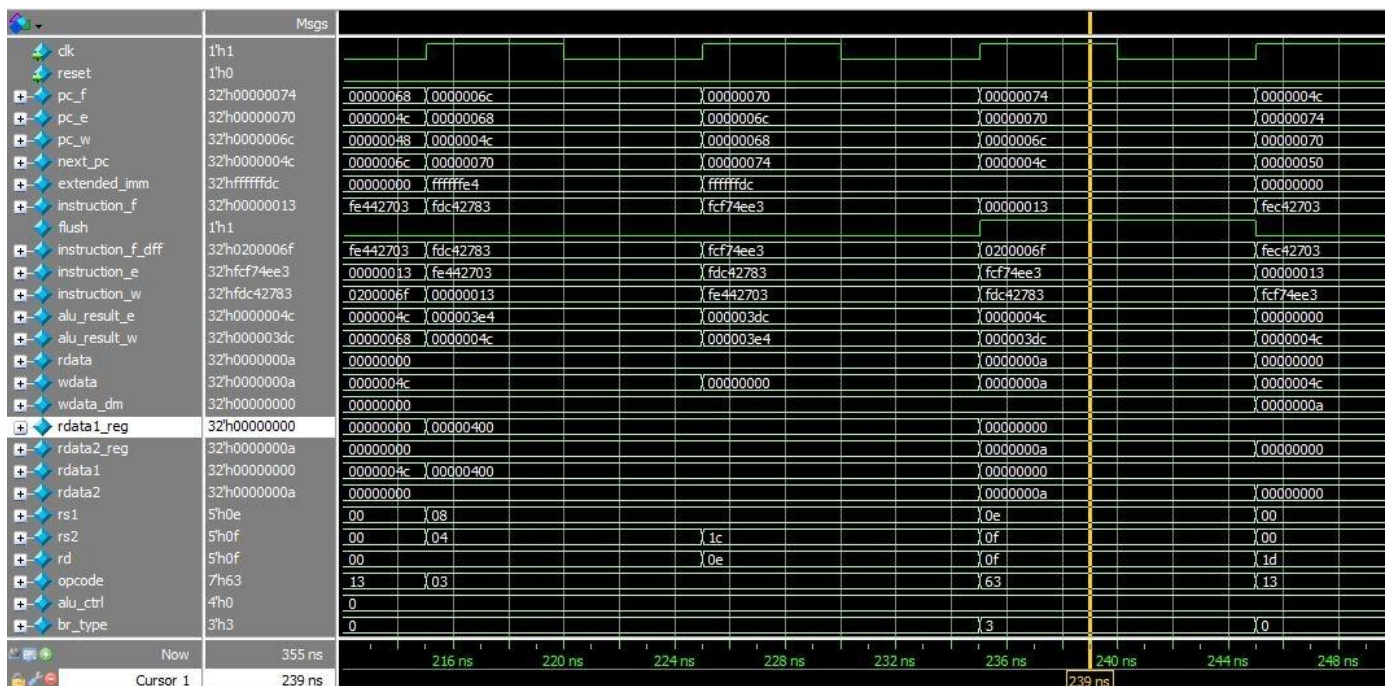
In this case, the forwarding stall unit again provides us with the correct value. It provides the data at the corresponding address of the data memory, instead of just providing us with the result of the ALU (which is just the address).

Control Hazard

Control Hazard occurs whenever we jump or take a branch. As the branch/jump instruction is in the execute phase, wrong instructions get loaded into the fetch stage. We don't want to execute those instructions as we are jumping or branching to some other point in code. So we flush the fetch stage to resolve this hazard. Let us execute the following instructions.

```
70:    fcf74ee3          blt    a4,a5,4c <main+0x4c>
74:    0200006f          j      94 <main+0x94>
```

Simulation

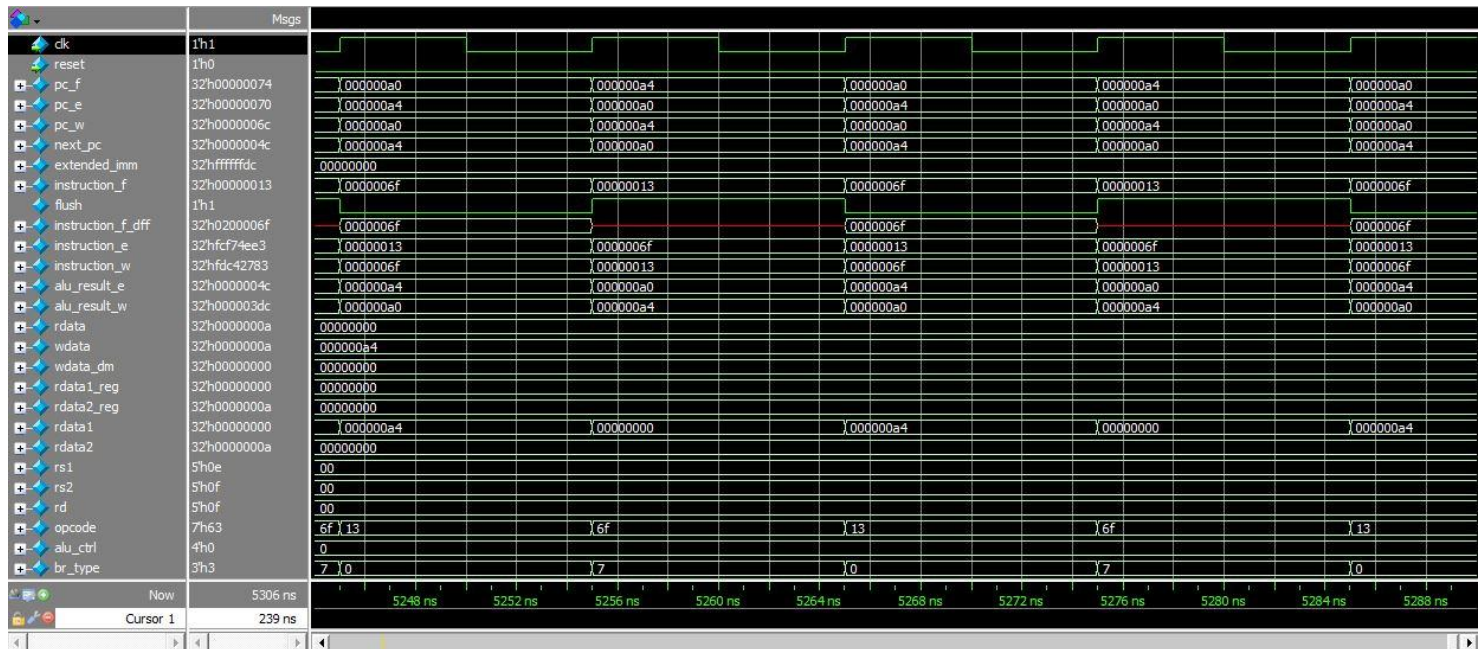


Here, we can observe that as the branch is taken, the fetch stage has been flushed. **Instruction_f** has value of **32'h00000013**, which is a **nop** instruction. This resolves the control hazard.

Let us observe the final output by executing this code.

```
24 00000000 <main>:
25 0: fd010113      addi sp,sp,-48
26 4: 02812623      sw s0,44(sp)
27 8: 03010413      addi s0,sp,48
28 c: 00500793      li a5,5
29 10: fef42023     sw a5,-32(s0)
30 14: 00a00793     li a5,10
31 18: fcf42e23     sw a5,-36(s0)
32 1c: fe042703     lw a4,-32(s0)
33 20: fdc42783     lw a5,-36(s0)
34 24: 00f707b3     add a5,a4,a5
35 28: fcf42c23     sw a5,-40(s0)
36 2c: fd842703     lw a4,-40(s0)
37 30: fe042783     lw a5,-32(s0)
38 34: 40f707b3     sub a5,a4,a5
39 38: fcf42a23     sw a5,-44(s0)
40 3c: fe042623     sw zero,-20(s0)
41 40: fe042423     sw zero,-24(s0)
42 44: fe042223     sw zero,-28(s0)
43 48: 0200006f     j 68 <main+0x68>
44 4c: fec42703     lw a4,-20(s0)
45 50: fd442783     lw a5,-44(s0)
46 54: 00f707b3     add a5,a4,a5
47 58: fef42623     sw a5,-20(s0)
48 5c: fe442783     lw a5,-28(s0)
49 60: 00178793     addi a5,a5,1
50 64: fef42223     sw a5,-28(s0)
51 68: fe442703     lw a4,-28(s0)
52 6c: fdc42783     lw a5,-36(s0)
53 70: fcf74ee3     blt a4,a5,4c <main+0x4c>
54 74: 0200006f     j 94 <main+0x94>
55 78: fec42703     lw a4,-20(s0)
56 7c: fe042783     lw a5,-32(s0)
57 80: 40f707b3     sub a5,a4,a5
58 84: fef42623     sw a5,-20(s0)
59 88: fe842783     lw a5,-24(s0)
60 8c: 00178793     addi a5,a5,1
61 90: fef42423     sw a5,-24(s0)
62 94: fec42703     lw a4,-20(s0)
63 98: fe042783     lw a5,-32(s0)
64 9c: fcf75ee3     bge a4,a5,78 <main+0x78>
65 a0: 0000006f     j a0 <main+0xa0>
66
```


Final Waveform



We can observe that the processor has entered the final instruction, and jumps back to it.

Memory Values

data - /pipeline_tb/dut/data_mem_0/memory - Default			
000003ff	XXXXXXXX	XXXXXXXX	XXXXXXXX 00000000
000003fb	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003f7	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003f3	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003ef	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003eb	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003e7	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003e3	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003df	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003db	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003d7	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003d3	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003cf	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003cb	XXXXXXXX	XXXXXXXX	XXXXXXXX
000003c7	XXXXXXXX	XXXXXXXX	XXXXXXXX