Pipelined ARM Processor

CIE 439 - Final Project

•••

Fall 2022

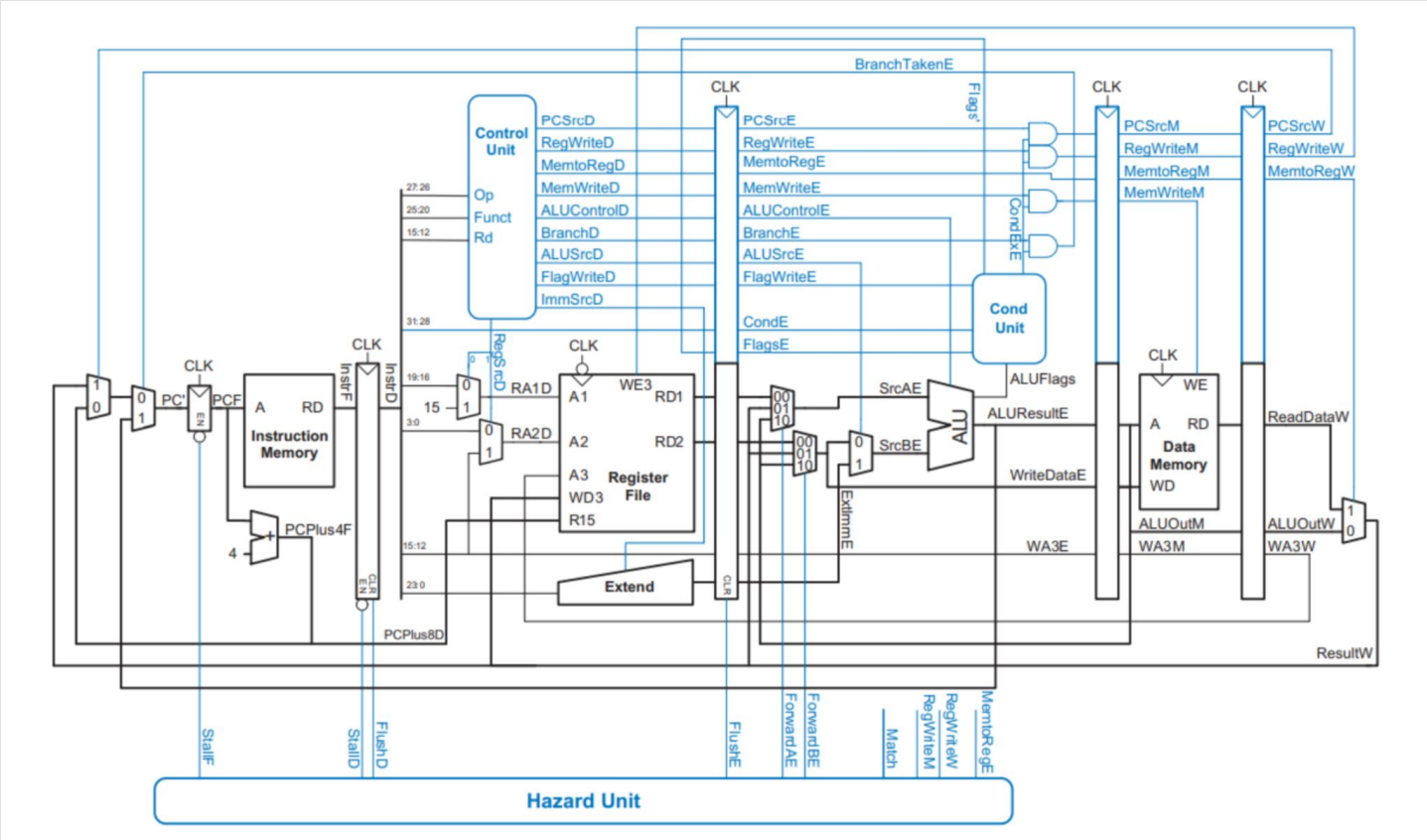
Our Team ✨

Amr Elmasry
201901202

Ahmed Ibrahem
201902227

Youssef Elshabrawy
201900667

Structure


BranchTakenE
CLK
Control Unit
Op
Funct
Rd
PCSrcD
RegWriteD
MemtoRegD
MemWriteD
ALUControlD
BranchD
ALUSrcD
FlagWriteD
ImmSrcD
PCSrcE
RegWriteE
MemtoRegE
MemWriteE
ALUControlE
BranchE
ALUSrcE
FlagWriteE
CondE
FlagsE
Flags
CondExE
Cond Unit
PCSrcM
RegWriteM
MemtoRegM
MemWriteM
PCSrcW
RegWriteW
MemtoRegW
PC'
PCF
Instruction Memory
InstF
InstD
RegSrcD
RA1D
RA2D
Register File
WE3
RD1
RD2
A1
A2
A3
WD3
R15
Extend
ImmExtE
SrcAE
SrcBE
ALU
ALUFlags
ALUResultE
WriteDataE
WA3E
Data Memory
WE
WD
ALUOutM
WA3M
ReadDataW
ALUOutW
WA3W
ResultW
PCPlus4F
PCPlus8D
StallF
StallD
FlushD
FlushE
ForwardAE
ForwardBE
Match
RegWriteM
RegWriteW
MemtoRegE
Hazard Unit

Test Bench

ADDR	PROGRAM	; COMMENTS	BINARY MACHINE CODE	HEX CODE
00 MAIN	SUB R0, R15, R15	; R0 = 0	1110 000 0010 0 1111 0000 0000 0000 1111	E04F000F
04	ADD R2, R0, #5	; R2 = 5	1110 001 0100 0 0000 0010 0000 0000 0101	E2802005
08	ADD R3, R0, #12	; R3 = 12	1110 001 0100 0 0000 0011 0000 0000 1100	E280300C
0C	SUB R7, R3, #9	; R7 = 3	1110 001 0010 0 0011 0111 0000 0000 1001	E2437009
10	ORR R4, R7, R2	; R4 = 3 OR 5 = 7	1110 000 1100 0 0111 0100 0000 0000 0010	E1874002
14	AND R5, R3, R4	; R5 = 12 AND 7 = 4	1110 000 0000 0 0011 0101 0000 0000 0100	E0035004
18	ADD R5, R5, R4	; R5 = 4 + 7 = 11	1110 000 0100 0 0101 0101 0000 0000 0100	E0855004
1C	SUBS R8, R5, R7	; R8 = 11 - 3 = 8, set Flags	1110 000 0010 1 0101 1000 0000 0000 0111	E0558007
20	BEQ END	; shouldn't be taken	0000 1010 0000 0000 0000 0000 0000 1100	0A00000C
24	SUBS R8, R3, R4	; R8 = 12 - 7 = 5	1110 000 0010 1 0011 1000 0000 0000 0100	E0538004
28	BGE AROUND	; should be taken	1010 1010 0000 0000 0000 0000 0000 0000	AA000000
2C	ADD R5, R0, #0	; should be skipped	1110 001 0100 0 0000 0101 0000 0000 0000	E2805000
30 AROUND	SUBS R8, R7, R2	; R8 = 3 - 5 = -2, set Flags	1110 000 0010 1 0111 1000 0000 0000 0010	E0578002
34	ADDLT R7, R5, #1	; R7 = 11 + 1 = 12	1011 001 0100 0 0101 0111 0000 0000 0001	B2857001
38	SUB R7, R7, R2	; R7 = 12 - 5 = 7	1110 000 0010 0 0111 0111 0000 0000 0010	E0477002
3C	STR R7, [R3, #84]	; mem[12+84] = 7	1110 010 1100 0 0011 0111 0000 0101 0100	E5837054
40	LDR R2, [R0, #96]	; R2 = mem[96] = 7	1110 010 1100 1 0000 0010 0000 0110 0000	E5902060
44	ADD R15, R15, R0	; PC = PC+8 (skips next)	1110 000 0100 0 1111 1111 0000 0000 0000	E08FF000
48	ADD R2, R0, #14	; shouldn't happen	1110 001 0100 0 0000 0010 0000 0000 0001	E280200E
4C	B END	; always taken	1110 1010 0000 0000 0000 0000 0000 0001	EA000001
50	ADD R2, R0, #13	; shouldn't happen	1110 001 0100 0 0000 0010 0000 0000 0001	E280200D
54	ADD R2, R0, #10	; shouldn't happen	1110 001 0100 0 0000 0010 0000 0000 0001	E280200A
58 END	STR R2, [R0, #100]	; mem[100] = 7	1110 010 1100 0 0000 0010 0000 0101 0100	E5802064

ALU OPTIMIZATION

ALU Optimization

For the BIC and EOR operations:

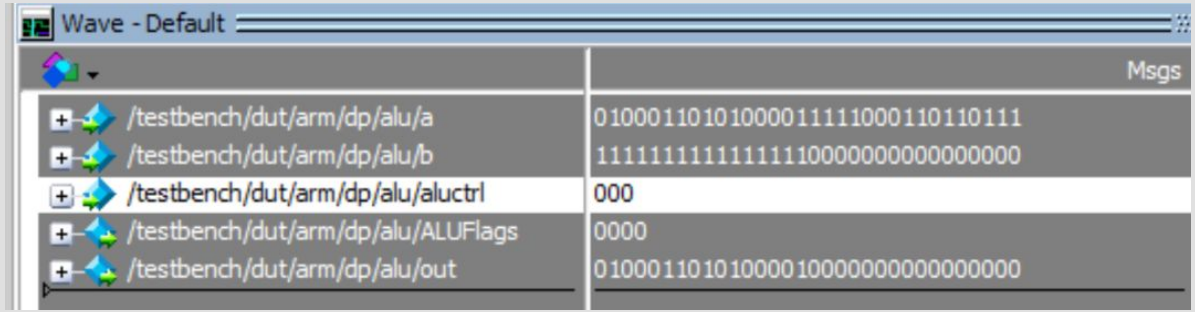
ALU is changed to support them

Operation	$ALUCtrl_2$	$ALUCtrl_1$	$ALUCtrl_0$
ADD	1	0	0
SUB	1	0	1
AND	0	0	0
ORR	0	0	1
XOR	0	1	0
BIC	0	1	1

Testing ALU

AND

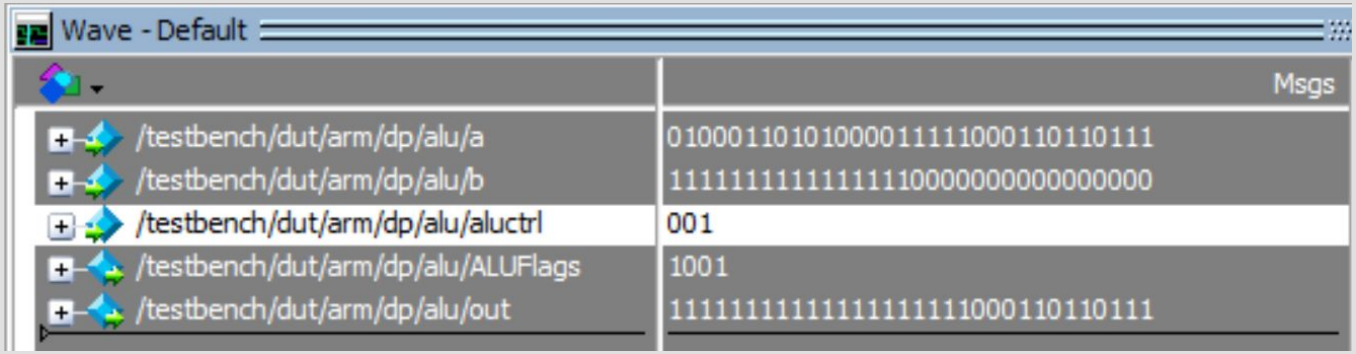
Operation	$ALUCtrl_2$	$ALUCtrl_1$	$ALUCtrl_0$
ADD	1	0	0
SUB	1	0	1
AND	0	0	0
ORR	0	0	1
XOR	0	1	0
BIC	0	1	1



Testing ALU

ORR

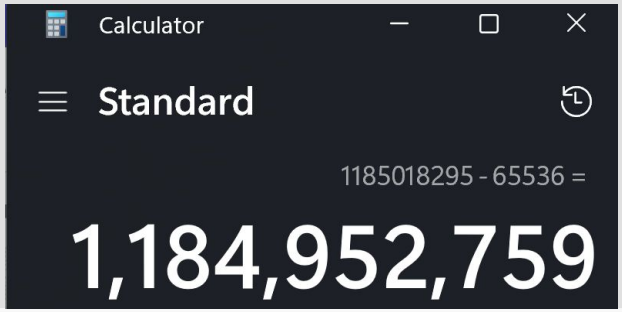
Operation	$ALUCtrl_2$	$ALUCtrl_1$	$ALUCtrl_0$
ADD	1	0	0
SUB	1	0	1
AND	0	0	0
ORR	0	0	1
XOR	0	1	0
BIC	0	1	1

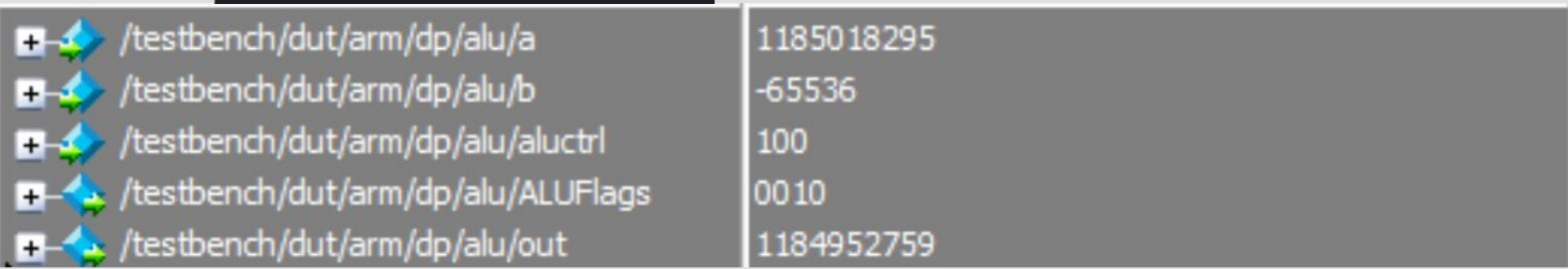


Testing ALU

ADD

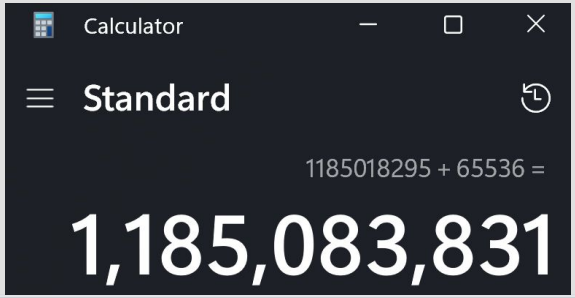
Operation	$ALUCtrl_2$	$ALUCtrl_1$	$ALUCtrl_0$
ADD	1	0	0
SUB	1	0	1
AND	0	0	0
ORR	0	0	1
XOR	0	1	0
BIC	0	1	1



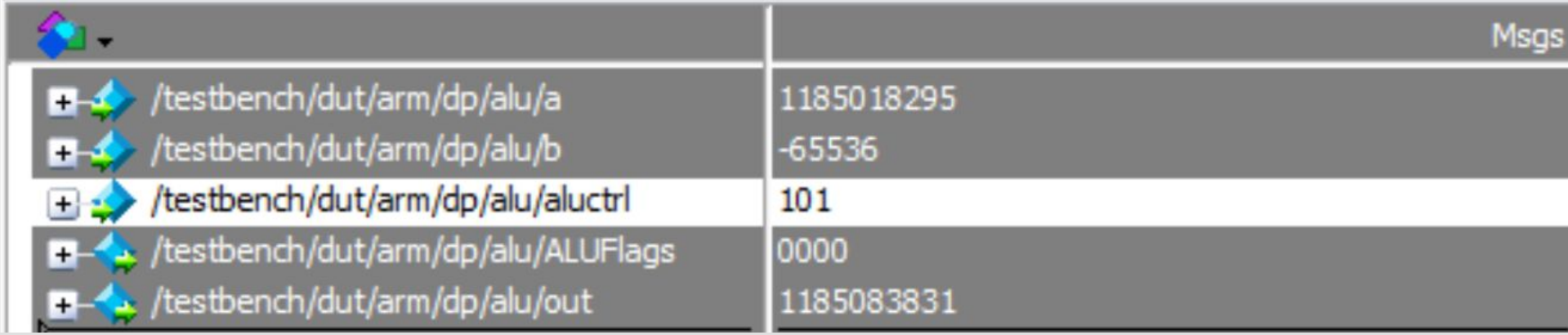


Testing ALU

SUB



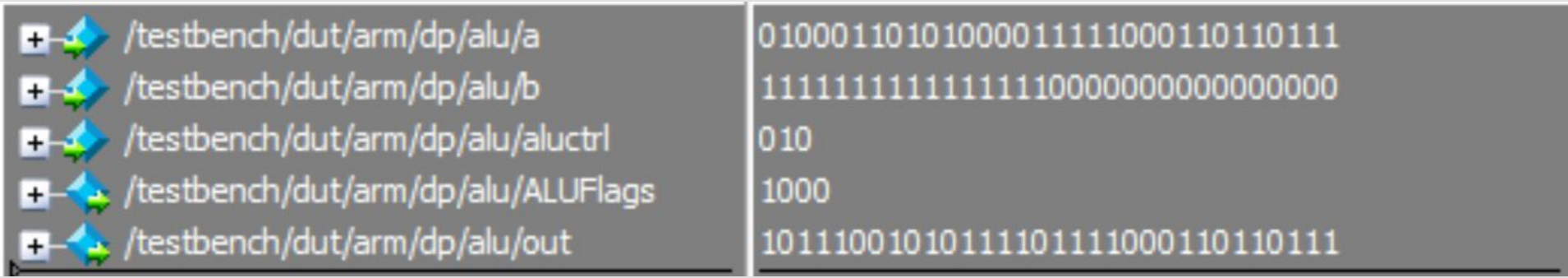
Operation	$ALUCtrl_2$	$ALUCtrl_1$	$ALUCtrl_0$
ADD	1	0	0
SUB	1	0	1
AND	0	0	0
ORR	0	0	1
XOR	0	1	0
BIC	0	1	1



Testing ALU

XOR

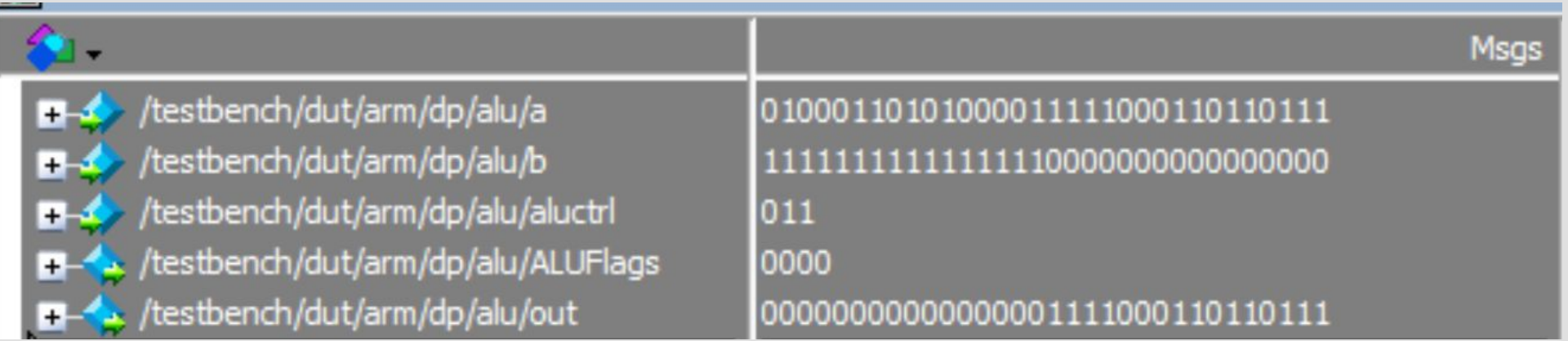
Operation	$ALUCtrl_2$	$ALUCtrl_1$	$ALUCtrl_0$
ADD	1	0	0
SUB	1	0	1
AND	0	0	0
ORR	0	0	1
XOR	0	1	0
BIC	0	1	1



Testing ALU

BIC

Operation	$ALUCtrl_2$	$ALUCtrl_1$	$ALUCtrl_0$
ADD	1	0	0
SUB	1	0	1
AND	0	0	0
ORR	0	0	1
XOR	0	1	0
BIC	0	1	1



Hazard Unit

Outline

1. Testing Read After Write Hazard (Data Hazard)
2. Testing Branch Hazards (Control Hazard)
3. Testing LDR Hazard (Data Hazard)

HAZARD UNIT

Testing Read After Write Hazard

1. E04F000F SUB R0,R15,R15
2. E2802005 ADD R2,R0,#5

   (ForwardAE is 10 at the 3rd cycle)
3. E280300C ADD R3,R0,#12

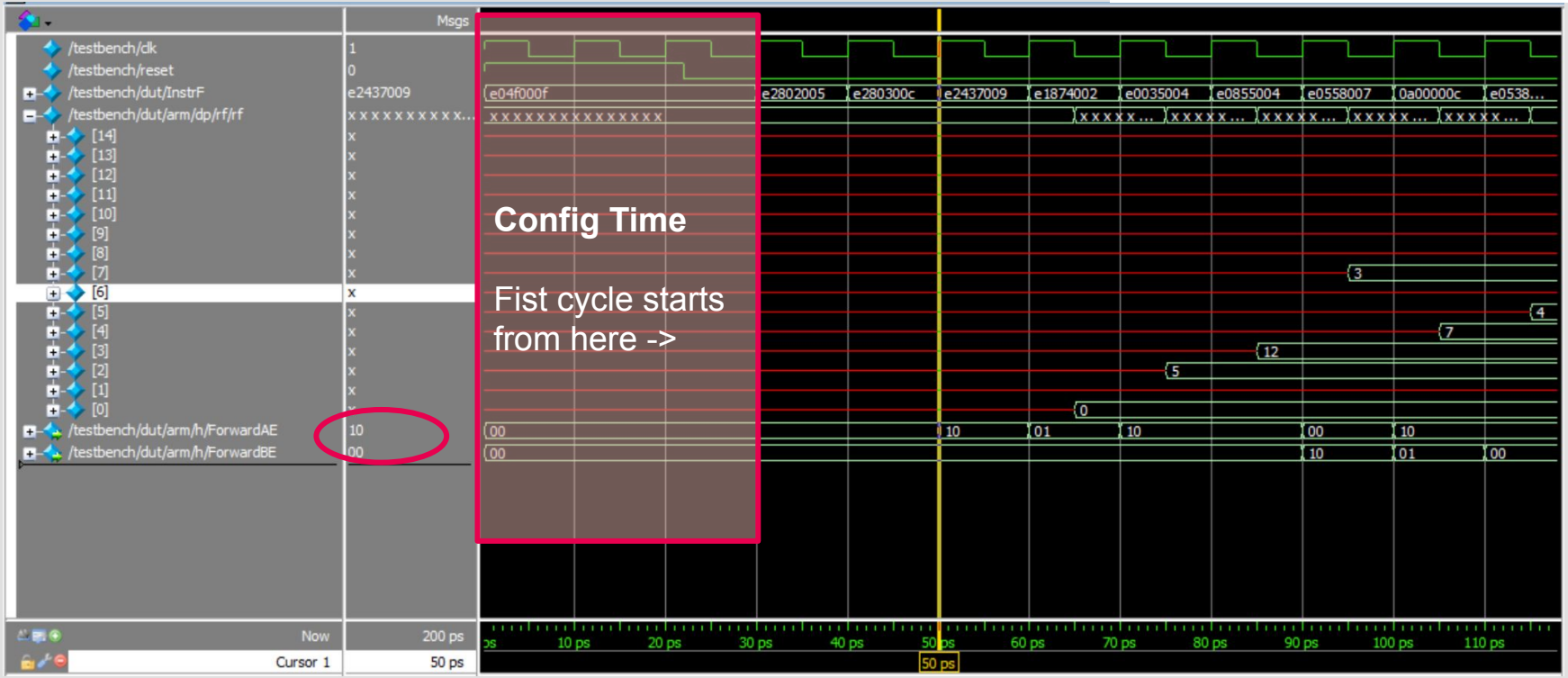
   (ForwardAE is 01 at the 4th cycle)

Testing read after write hazard

@Cycle 3: ForwardAE = 10 (working!)

Testing Read After Write Hazard

1. E04F000F SUB R0,R15,R15
2. E2802005 ADD R2,R0,#5
   (ForwardAE is 10 at the 3rh cycle)
3. E280300C ADD R3,R0,#12
   (ForwardAE is 01 at the 4th cycle)

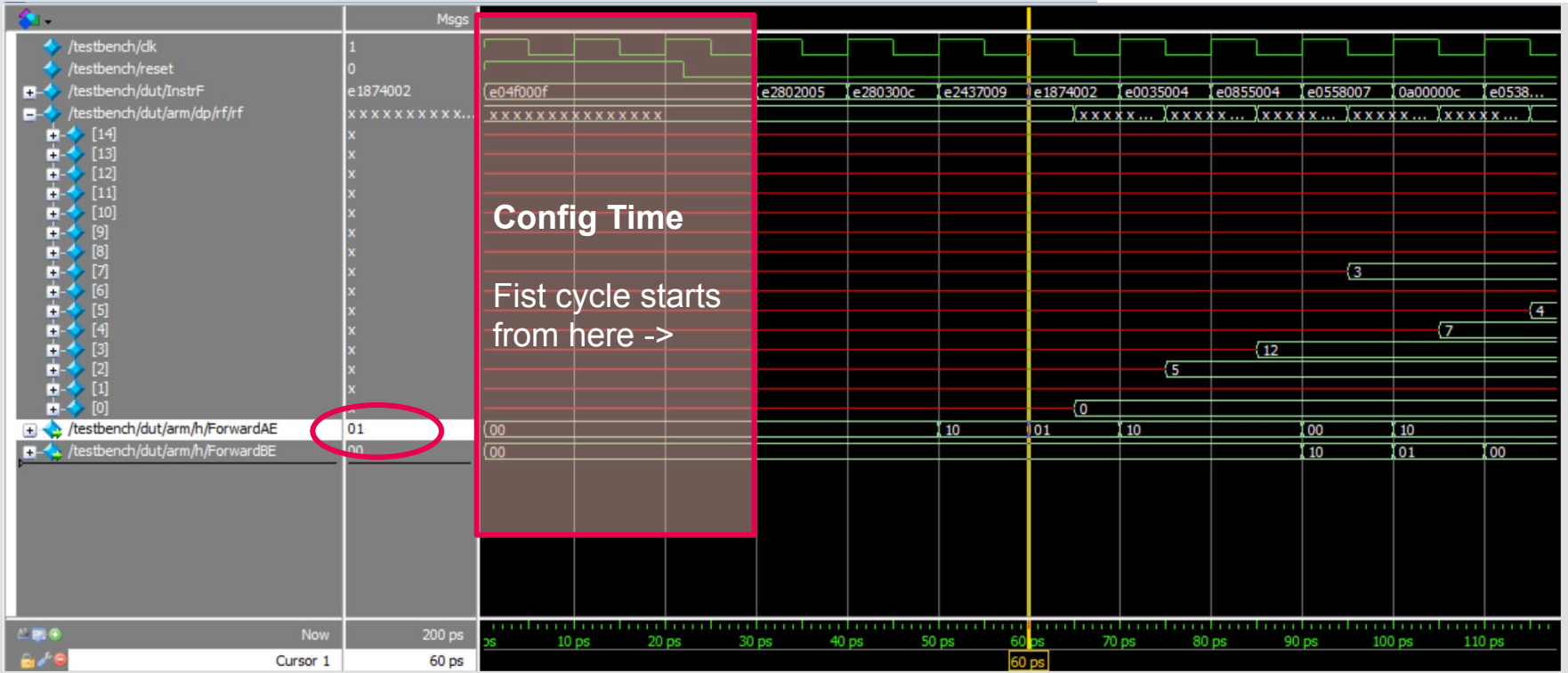


Testing read after write hazard

@Cycle 4: ForwardAE = 01 (working!)

Testing Read After Write Hazard

1. E04F000F SUB R0,R15,R15
2. E2802005 ADD R2,R0,#5
   (ForwardAE is 10 at the 3rh cycle)
3. E280300C ADD R3,R0,#12
   (ForwardAE is 01 at the 4th cycle)

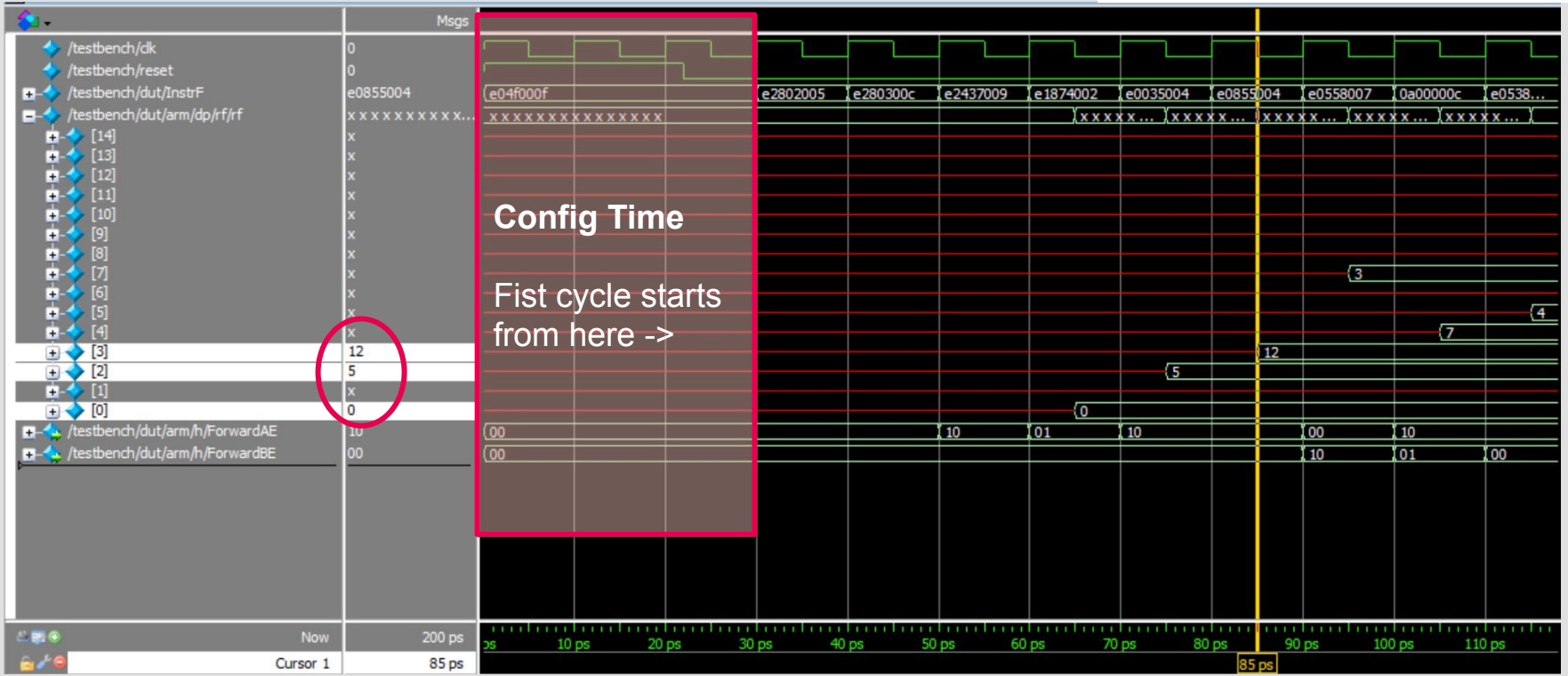


Testing read after write hazard

@ Cycle 6(second half): R2 and R3 are updated with the right values

Testing Read After Write Hazard

1. E04F000F SUB R0,R15,R15
2. E2802005 ADD R2,R0,#5
   (ForwardAE is 10 at the 3rh cycle)
3. E280300C ADD R3,R0,#12
   (ForwardAE is 01 at the 4th cycle)



HAZARD UNIT

Read After Write Hazards

Done ✅

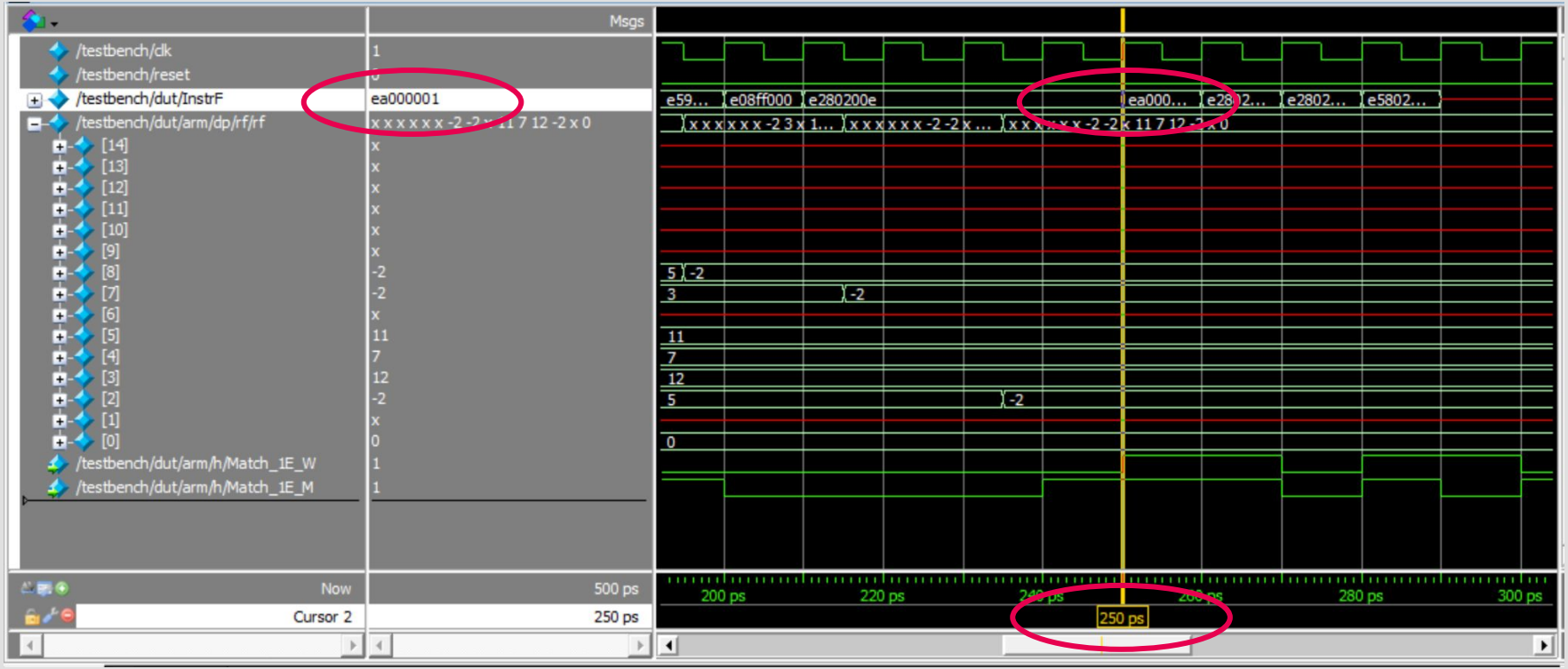
HAZARD UNIT

Testing Branch Hazards

20. EA000001 B END
21. E280200D ADD R2,R0, #13 (Should NOT happen)
22. E280200A ADD R2, R0, #10 (Should NOT happen)

Testing Branch Hazards

First: To find when the instruction is ready we search in the InstrF

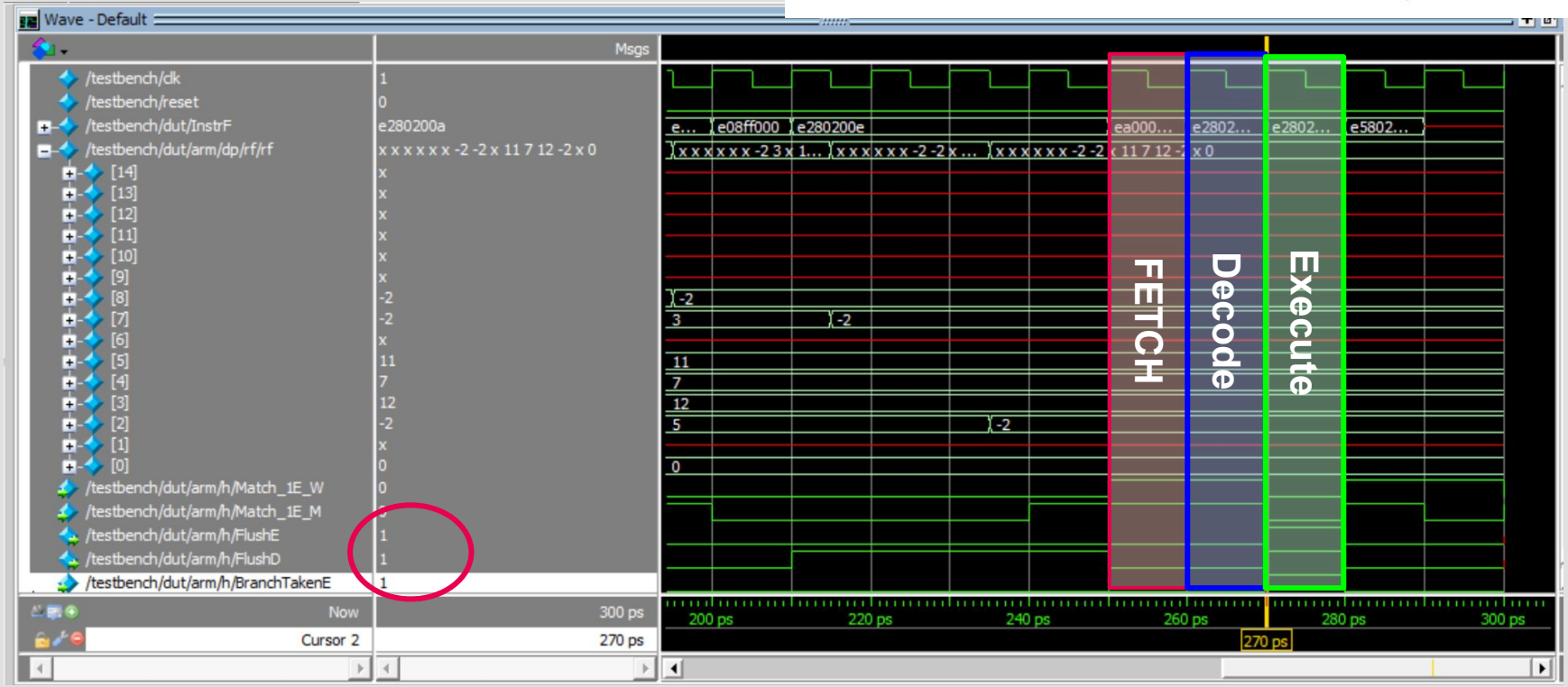


Testing Branch Hazards

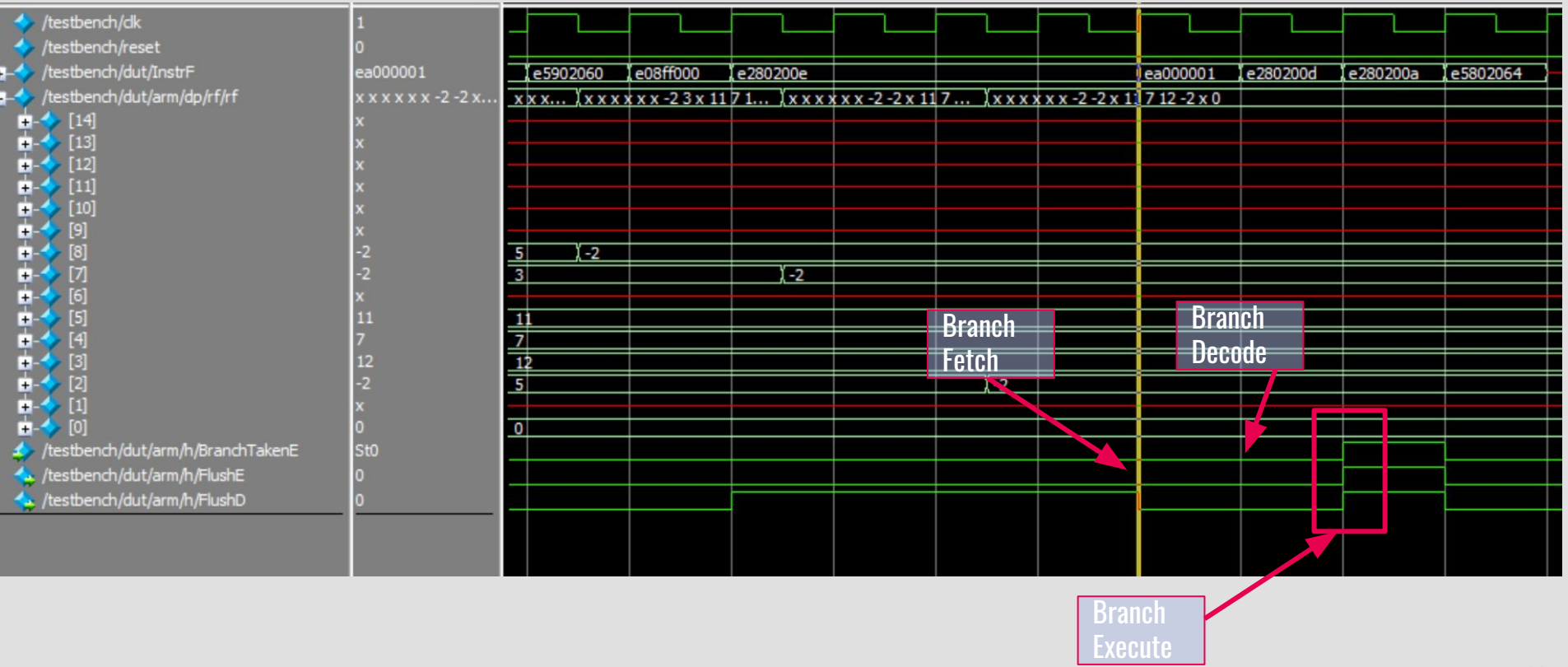
@ **Branch Execute:** Flush signals are immediately fired to delete the instructions before them getting executed

Testing Branch Hazards

20.	EA000001	B END
21.	E280200D	ADD R2,R0, #13 (Should NOT happen)
22.	E280200A	ADD R2, R0, #10 (Should NOT happen)



ZOOM IN 🔍

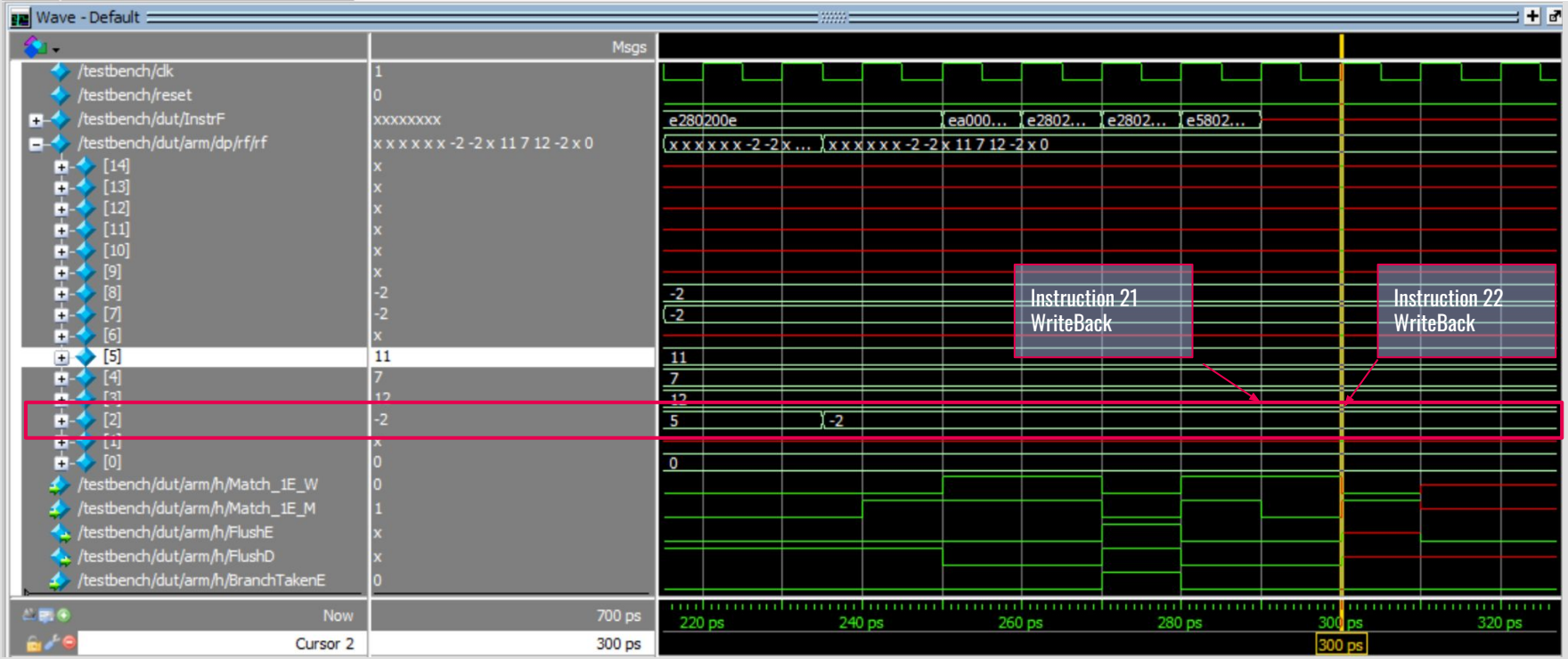

/testbench/clk
/testbench/reset
/testbench/dut/InstrF
/testbench/dut/arm/dp/rf/rf
[14]
[13]
[12]
[11]
[10]
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/testbench/dut/arm/h/BranchTakenE
/testbench/dut/arm/h/FlushE
/testbench/dut/arm/h/FlushD
1
0
ea000001
x x x x x x -2 -2 x...
e5902060
e08ff000
e280200e
ea000001
e280200d
e280200a
e5802064
Branch Fetch
Branch Decode
Branch Execute

Testing Branch Hazards

After Branch: R2 value never changed as expected

Testing Branch Hazards

20. EA000001 B END
21. E280200D ADD R2,R0, #13 (Should NOT happen)
22. E280200A ADD R2, R0, #10 (Should NOT happen)



HAZARD UNIT

Branch Hazards

Done ✅

HAZARD UNIT

Testing LDR Hazard

We should prepare a testbench for this case

1. SUB R4, R15, R15
2. SUB R5, R15, R15
3. STR R5,[R5,#64]
4. LDR R1, [R4,#64]
5. ORR R7,R1,R5

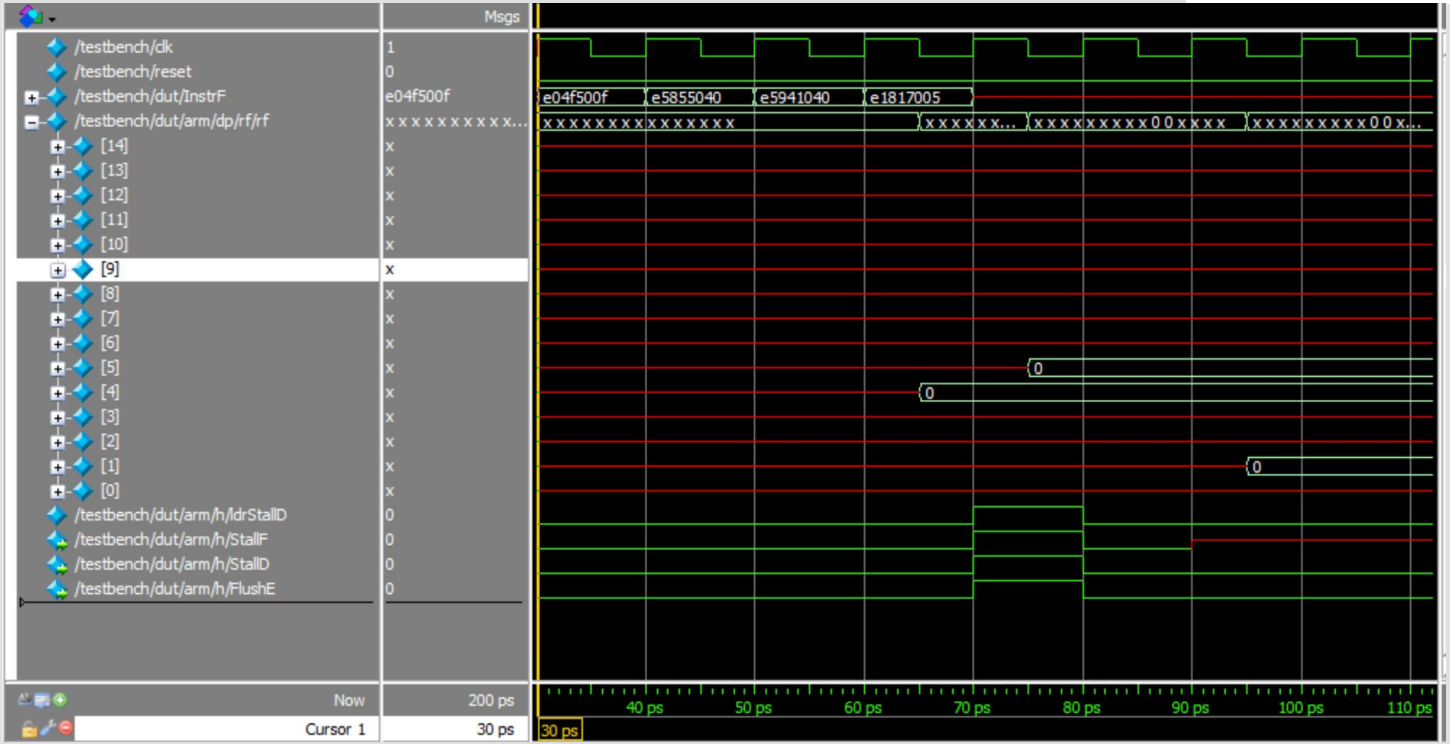
New Test Bench

ADDR	CODE	BINARY	HEX
00	SUB R4, R15, R15 ; R4 = 0	1110 000 0010 0 1111 0100 0000 0000 1111	E04F400F
04	SUB R5, R15, R15 ; R5 = 0	1110 000 0010 0 1111 0101 0000 0000 1111	E04F500F
08	STR R5,[R5,#64] ; Address has content = 0	1110 0101 1000 0101 0101 0000 0100 00000	E5855040
0C	LDR **R1**, [R4,#64] ; Load 0 in R1	1110 0101 1001 0100 0001 0000 0100 00000	E5941040
14	ORR R7,**R1**,R5 ; Perform 0 ORR 0 ; R1 should have 0	1110 0001 1000 0001 0111 0000 0000 00101	E1817005

Testing LDR Hazard

@C1: I1 is Fetched

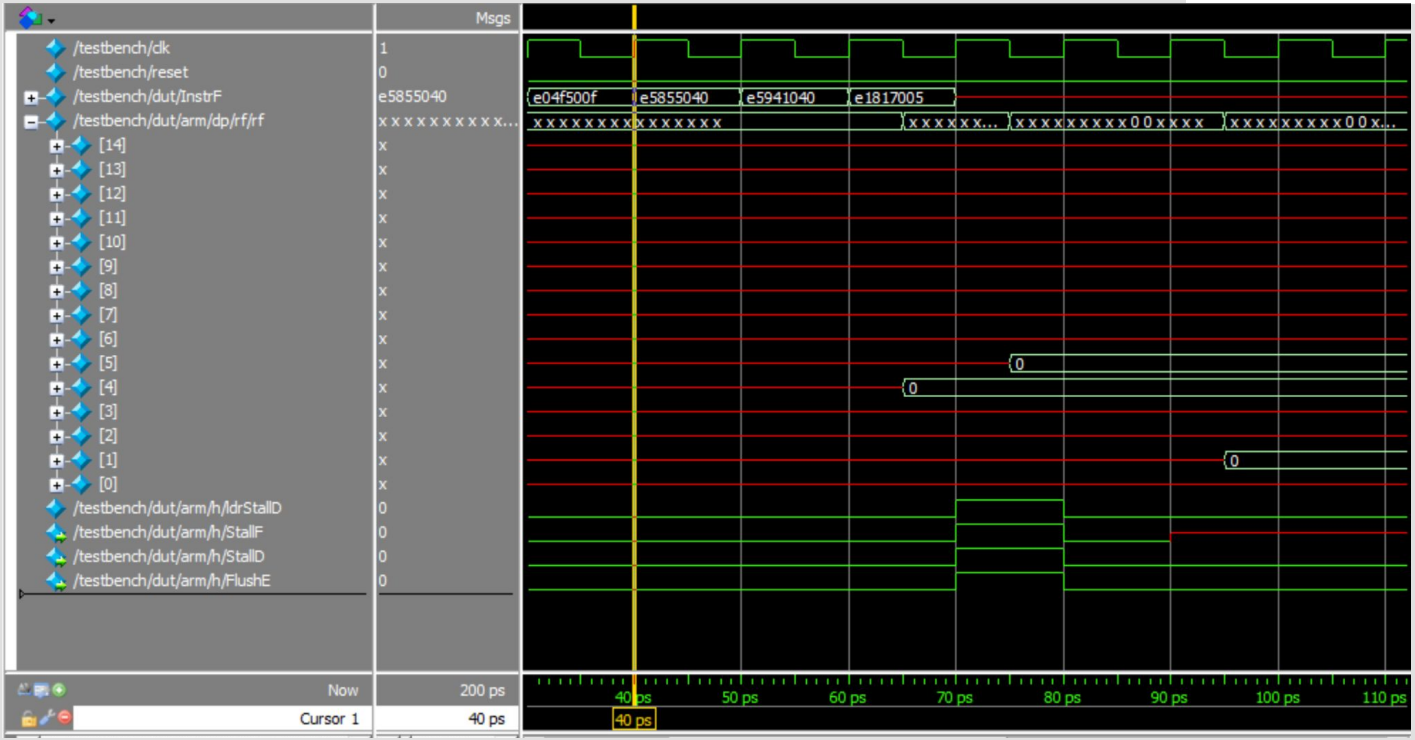
1. E04F400F SUB R4, R15, R15
2. E04F500F SUB R5, R15, R15
3. E5855040 STR R5,[R5,#64]
4. E5941040 LDR R1, [R4,#64]
5. E1817005 ORR R7,R1,R5



Testing LDR Hazard

@C2: I2 is Fetched, I1 is decoded

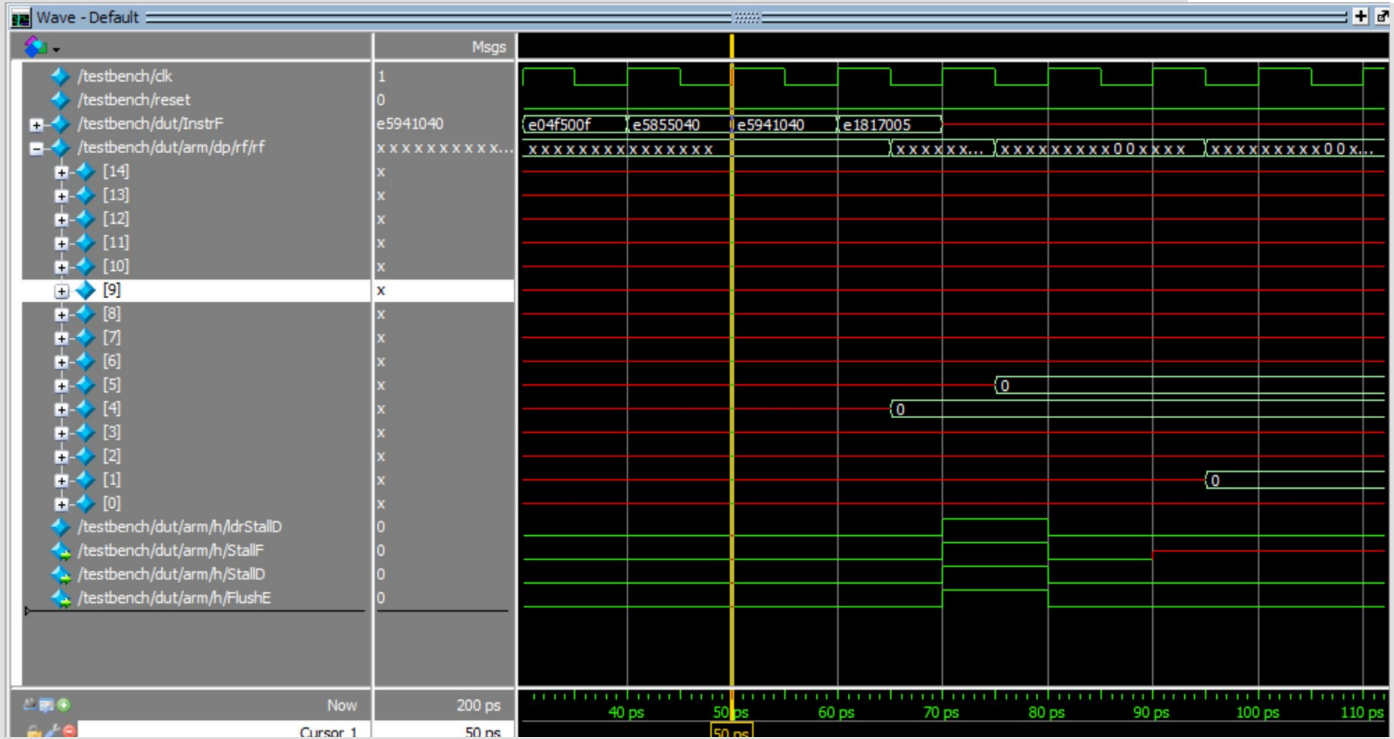
1.	E04F400F	SUB R4, R15, R15
2.	E04F500F	SUB R5, R15, R15
3.	E5855040	STR R5,[R5,#64]
4.	E5941040	LDR R1, [R4,#64]
5.	E1817005	ORR R7,R1,R5



Testing LDR Hazard

@C3: I3 is Fetched, I2 is decoded, I1 is executed

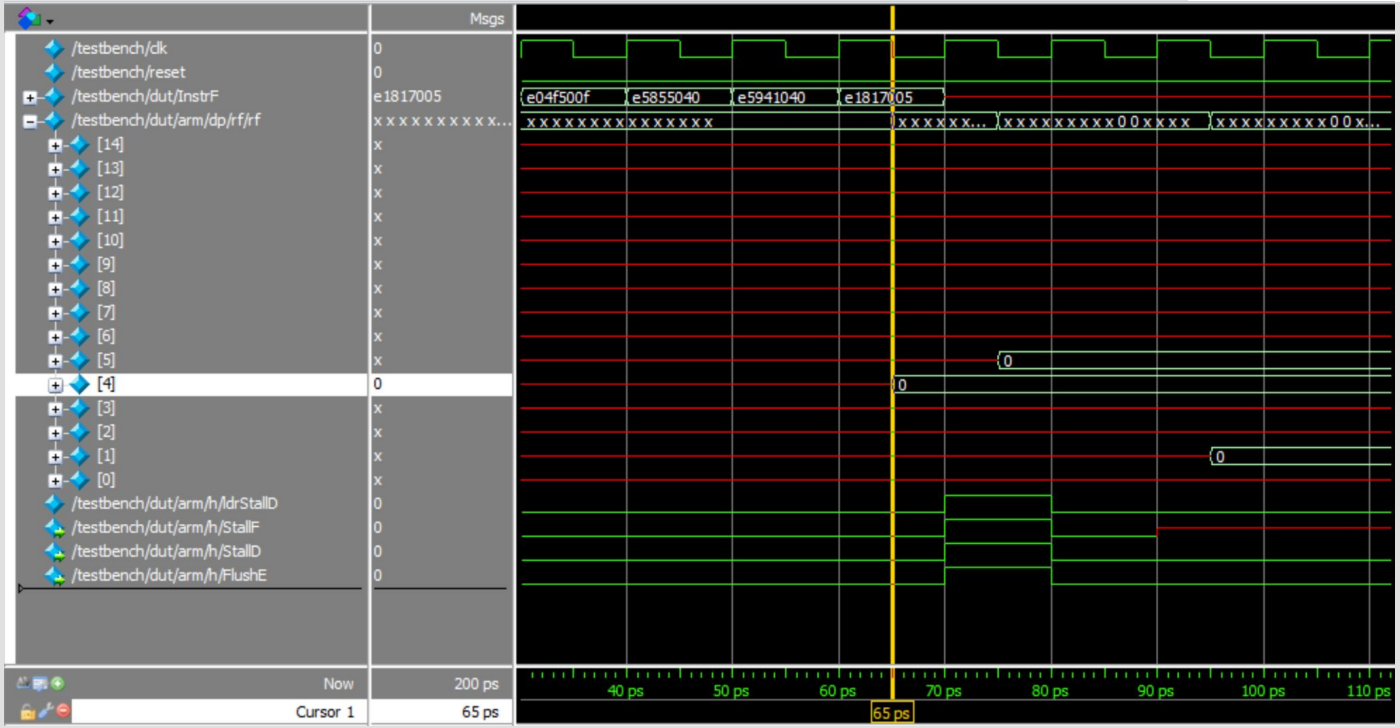
1. E04F400F SUB R4, R15, R15
2. E04F500F SUB R5, R15, R15
3. E5855040 STR R5,[R5,#64]
4. E5941040 LDR R1, [R4,#64]
5. E1817005 ORR R7,R1,R5



Testing LDR Hazard

@C4: I4 is Fetched, I3 is decoded, I2 is executed, I1 is WB (2nd half)

1. E04F400F SUB R4, R15, R15
2. E04F500F SUB R5, R15, R15
3. E5855040 STR R5,[R5,#64]
4. E5941040 LDR R1, [R4,#64]
5. E1817005 ORR R7,R1,R5



R4 = 0

Testing LDR Hazard

@C5: I5 is Fetched, **I4 is decoded**, I3 is executed, I2 is WB (2nd half)

1. E04F400F SUB R4, R15, R15
2. E04F500F SUB R5, R15, R15
3. E5855040 STR R5,[R5,#64]
4. E5941040 LDR R1, [R4,#64]
5. E1817005 ORR R7,R1,R5

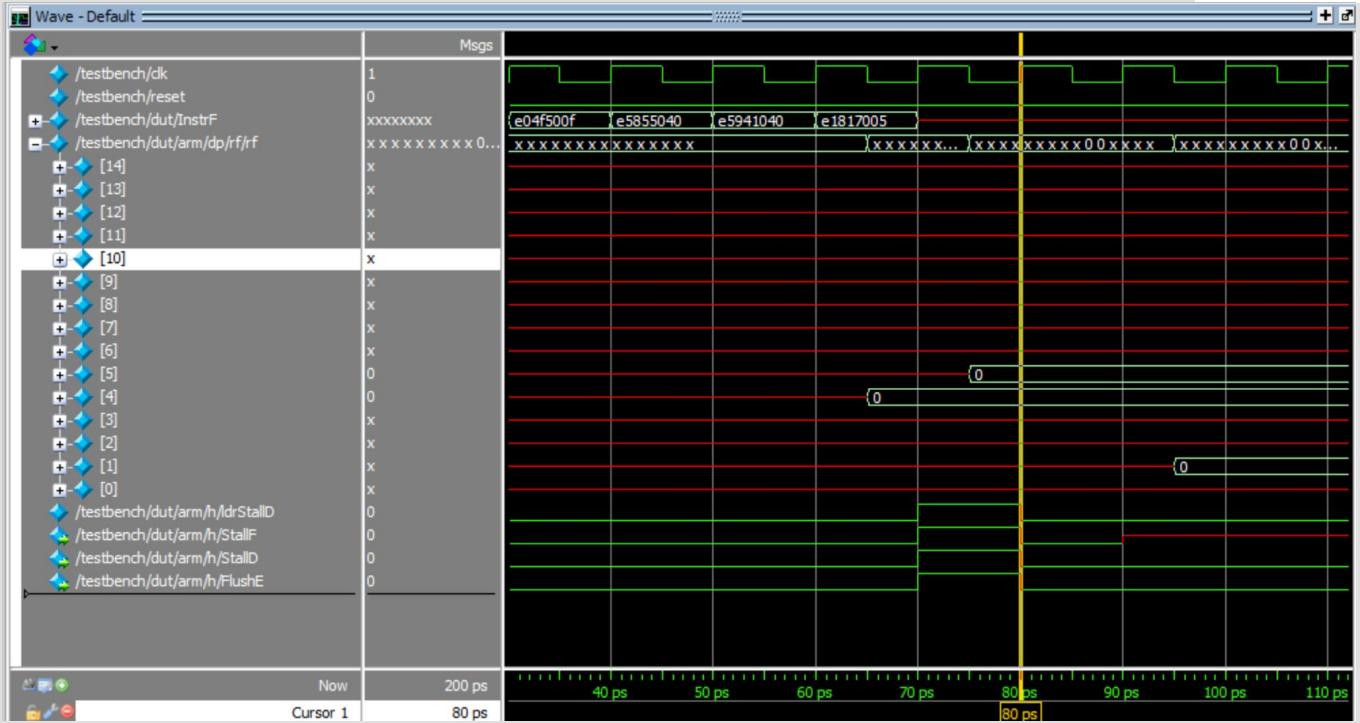
R5 = 0

Stall and Flush!

Testing LDR Hazard

@C6: I5 is decoded, I4 is executed, I3 is WB (2nd half)

1. E04F400F SUB R4, R15, R15
2. E04F500F SUB R5, R15, R15
3. E5855040 STR R5,[R5,#64]
4. E5941040 LDR R1, [R4,#64]
5. E1817005 ORR R7,R1,R5



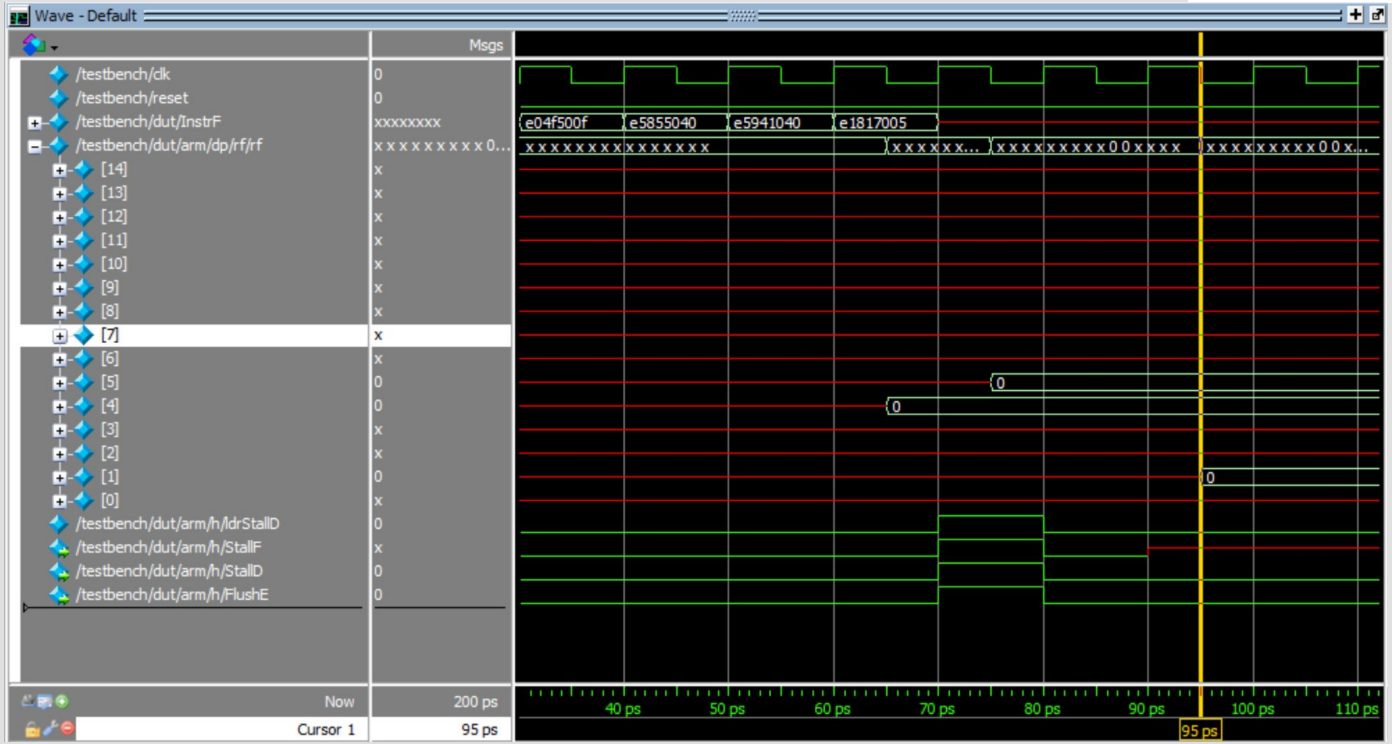
STALL IS HAPPENING

[R5,#64] has 0

Testing LDR Hazard

@C7: I5 is executed, I4 is WB (2nd half)

1. E04F400F SUB R4, R15, R15
2. E04F500F SUB R5, R15, R15
3. E5855040 STR R5,[R5,#64]
4. E5941040 LDR R1, [R4,#64]
5. E1817005 ORR R7,R1,R5

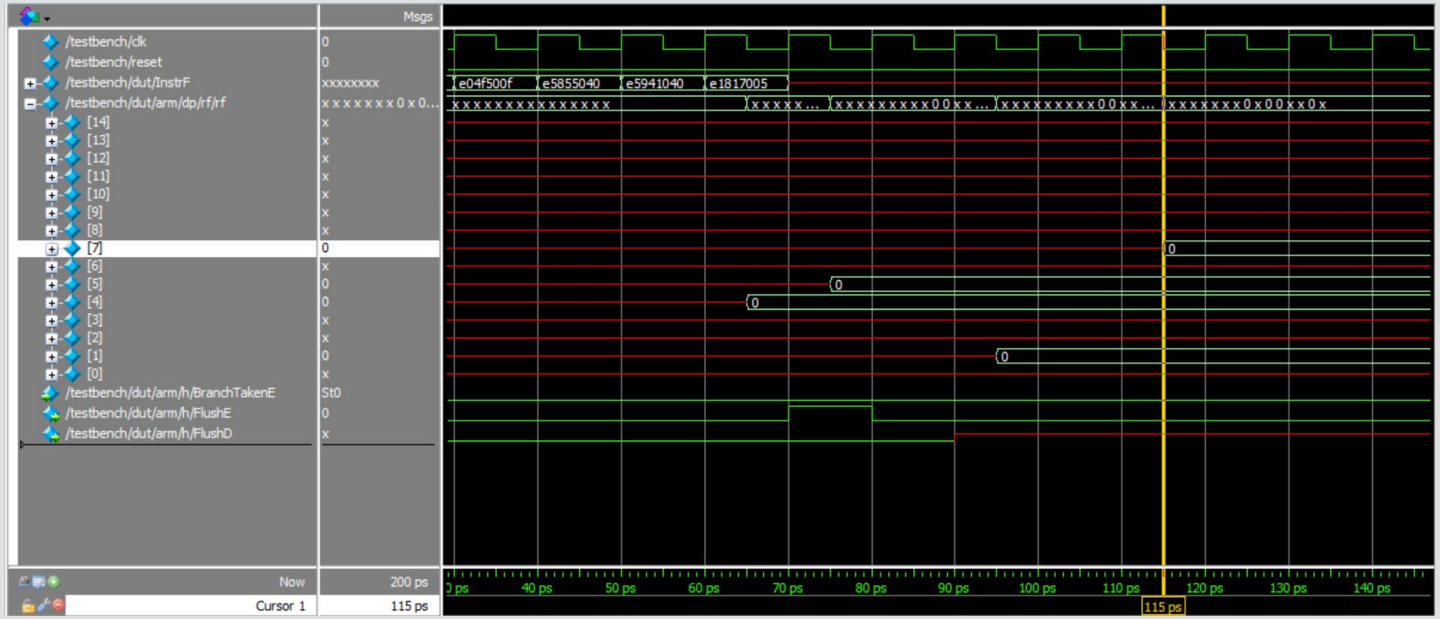


R1 = 0

Testing LDR Hazard

@C8: I5 is WB (2nd half)

```
1.  E04F400F    SUB R4, R15, R15
2.  E04F500F    SUB R5, R15, R15
3.  E5855040    STR R5,[R5,#64]
4.  E5941040    LDR R1, [R4,#64]
5.  E1817005    ORR R7,R1,R5
```



R7 = 0

Without stall R7 would have been X

HAZARD UNIT

LDR Hazards

Done ✅

Thank You ✨