Architecture Final Assessment

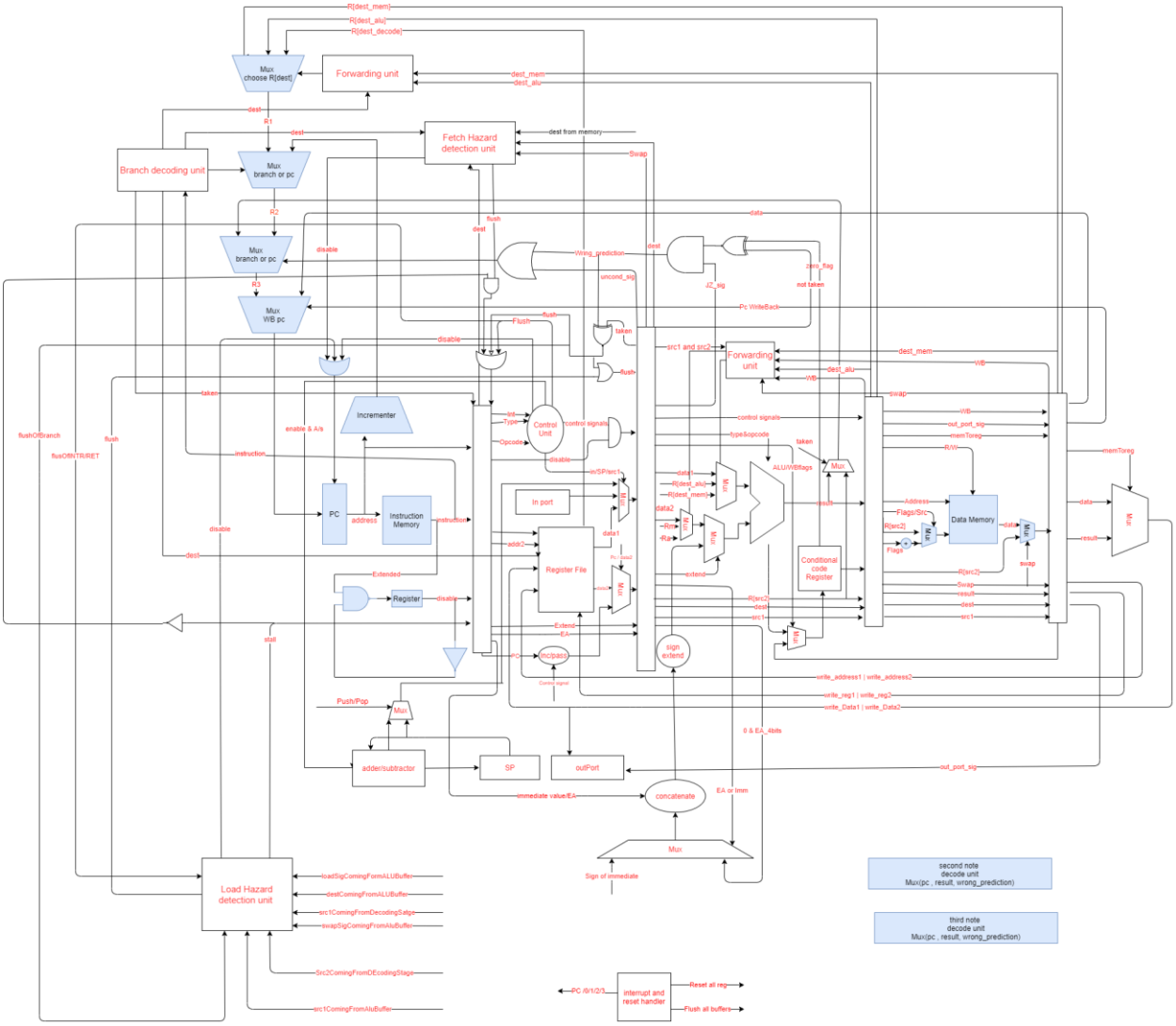
Submitted by:

Name	Sec	BN
Ahmed Essam Eldeen	1	3
Philopateer Nabil Atia	2	4
Mazen Amr Fawzy	2	8
Mahmoud Ahmed Sebak	2	20

Table of Contents

Processor Design

Schematic link for better resolution:
https://drive.google.com/file/d/1zErDoxThT1pvpNCppryKj9otKL1R4Lfs/view?usp=sharing

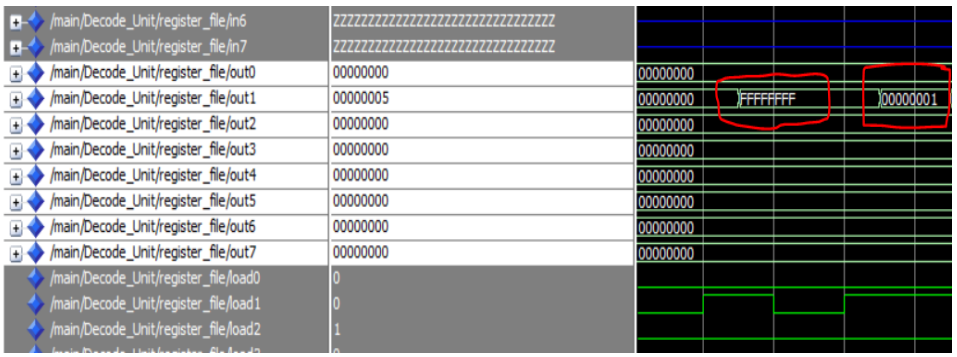


One operand

```
1  NOT R1          #R1 =FFFFFFFF , C--> no change, N --> 1, Z --> 0
2  NOP             #No change
3  inc R1          #R1 =00000000 , C --> 1 , N --> 0 , Z --> 1
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= FFFFFFEF, C--> no change, N -->1,Z-->0
7  inc R1          #R1= 6, C --> 0, N -->0, Z-->0
8  Dec R2          #R2= FFFFFFEE,C-->1 , N-->1, Z-->0
9  out R1
10 out R2
```

(Fig. 1)

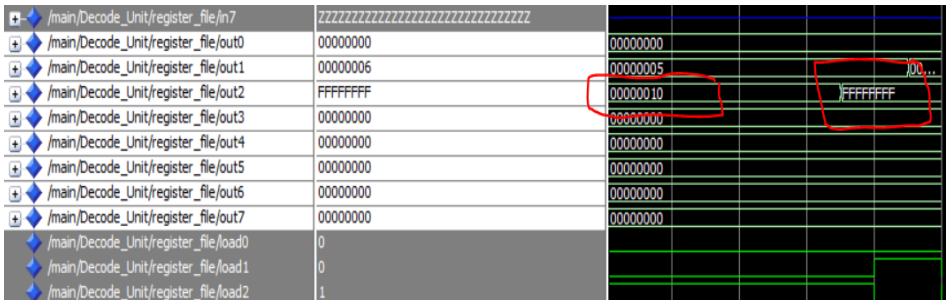
1. Without forwarding units and hazard detection units
   - **Hazards detected:**
     - Read after write hazard detected between instruction 1 and instruction 3 (Look Fig. 1).



(Fig. 2)

As shown in (Fig. 2) (a screenshot from our simulation wave), R1 (out1) was incremented on the old value (00000000) not the new one (FFFFFFFF) so it became 00000001, however the correct result is 00000000.

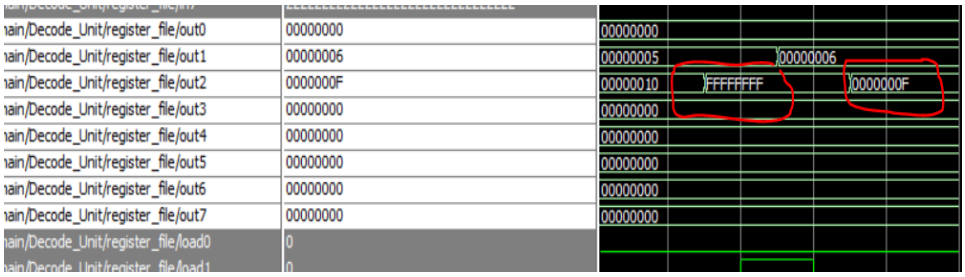
- Read after write hazard detected between instruction 5 and instruction 6 (Look Fig. 1).



. (Fig. 3)

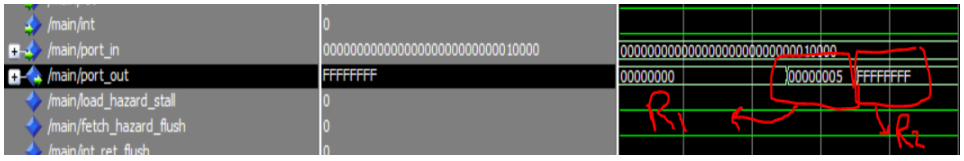
As shown (Fig. 3) R2 is “FFFFFFFF” which is the not operation of “00000000” (the old value), however the correct answer is “FFFFFFEF” which is the not operation of (“00000010”) which is taken from the input port.

- Read after write hazard detected between instruction 6 and instruction 8 (Look Fig. 1)



. (Fig. 4)

- Read after write hazard detected between instruction 8 and instruction 10 (Look Fig. 1)
- Read after write hazard detected between instruction 7 and instruction 9 (Look Fig. 1)



(Fig. 5)

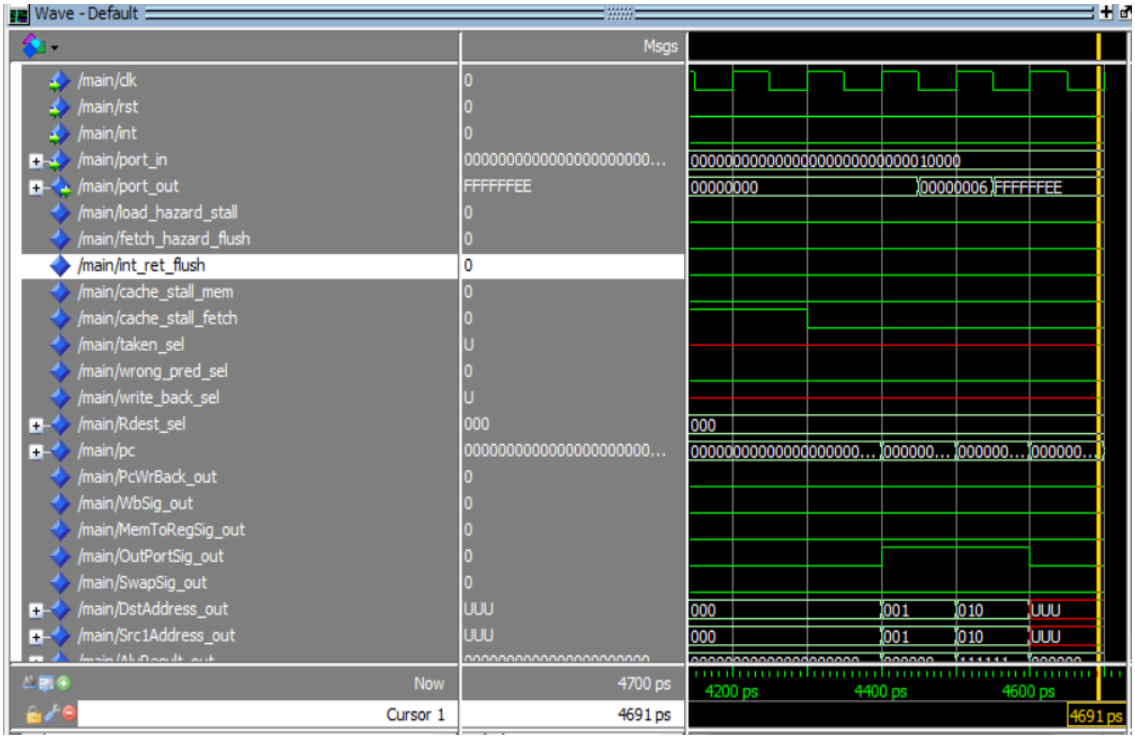
As shown in (Fig. 5), the out port contains wrong values of R1, and R2. The correct values should have been “00000006” and “FFFFFFE”. This happened because of a lot of read after write hazards.

- **Solving the hazards by inserting “nop” in the code**

```
# all numbers in hex format
# we always start by reset signal
#this is a commented line
.ORG 0  #this means the the following line would be  at address  0 , and this is the reset address
10
#you should ignore empty lines

.ORG 2  #this is the interrupt address
100

.ORG 10
NOT R1          #R1 =FFFFFFFF , C--> no change, N --> 1, Z --> 0
NOP             #No change
Nop
inc R1          #R1 =00000000 , C --> 1 , N --> 0 , Z --> 1
in R1           #R1= 5,add 5 on the in port,flags no change
in R2           #R2= 10,add 10 on the in port, flags no change
nop
nop
NOT R2          #R2= FFFFFFEF, C--> no change, N -->1,Z-->0
inc R1
nop          #R1= 6, C --> 0, N -->0, Z-->0
Dec R2          #R2= FFFFFFEE,C-->1 , N-->1, Z-->0
nop
out R1
out R2

```

(Fig. 6)

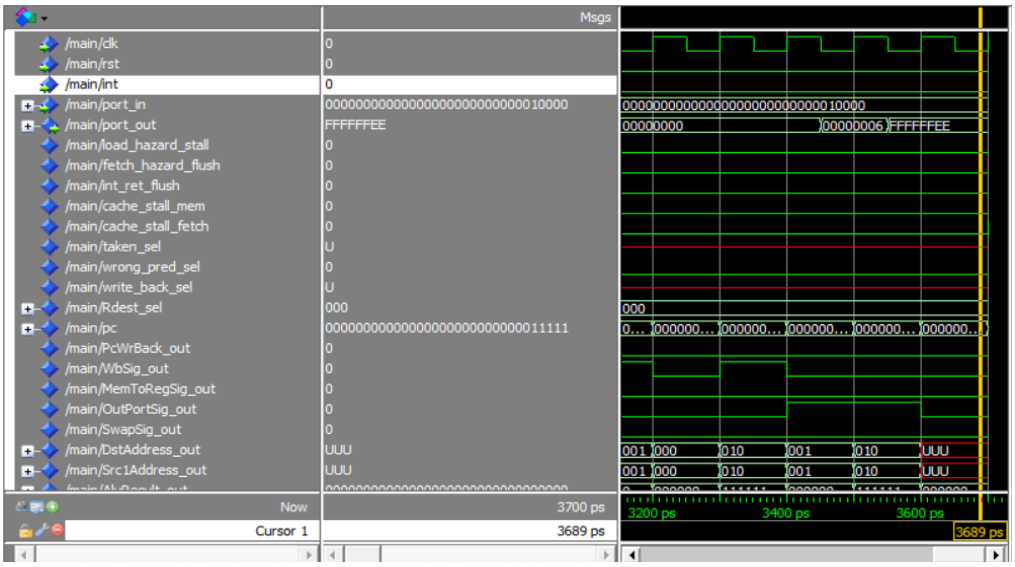
As shown in (Fig. 6), we inserted no operations between any dependencies to prevent hazards. The output port contains the right values after these modifications (look Fig. 7).



(Fig. 7)

2. With the forwarding units added

By adding the forwarding units there is no hazards and the output was correct (look Fig. 8)



(Fig. 8)

3. Speed up due to adding forwarding units

As shown in (Fig. 7), in case of no forwarding units and handling hazards by inserting no operations, the program is executed in 4600 ps.

As shown in (Fig. 8), after adding all forwarding units, the program is executed in 3600 ps.

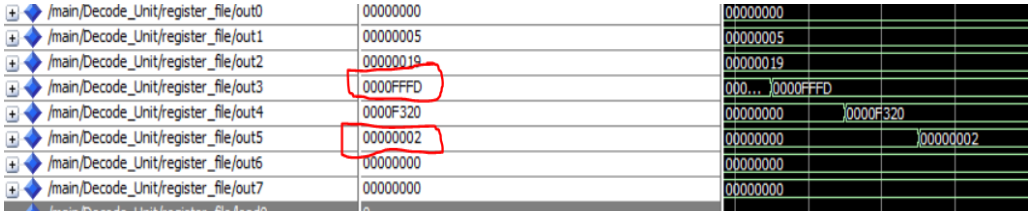
So the speed up equals (4600/3600) 1.278.

Two operands

```
1  in   R1        #add 5 in R1
2  in   R2        #add 19 in R2
3  in   R3        #FFFD
4  in   R4        #F320
5  IADD R3,R5,2   #R5 = FFFF , flags no change
6  ADD  R1,R4,R4     #R4= F325 , C-->0, N-->0, Z-->0
7  SUB  R5,R4,R6     #R6= 0CDA , C-->1, N-->0,Z-->0
8  AND  R7,R6,R6     #R6= 00000000 , C-->no change, N-->0, Z-->1
9  OR   R2,R1,R1     #R1=1D  , C--> no change, N-->0, Z--> 0
10 SHL  R2,2      #R2=64  , C--> 0, N -->0 , Z -->0
11 SHR  R2,3      #R2=0C  , C -->1, N-->0 , Z-->0
12 SWAP R2,R5     #R5=0C ,R2=FFFF  ,no change for flags
13 ADD  R5,R2,R2     #R2= 1000B (C,N,Z= 0)
```

(Fig. 10)

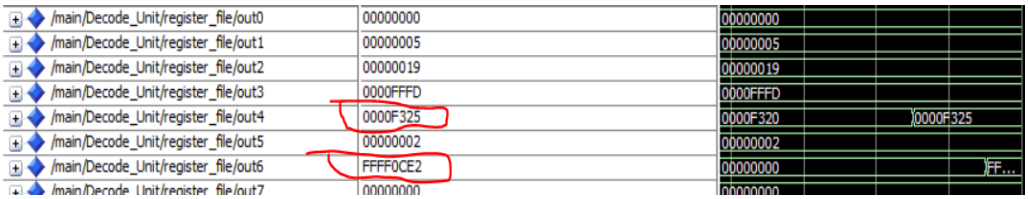
1. Without forwarding units
    - **Hazards detected**
        - Read after write hazard detected between instruction 3 and instruction 5 (see Fig. 10)



(Fig. 11)

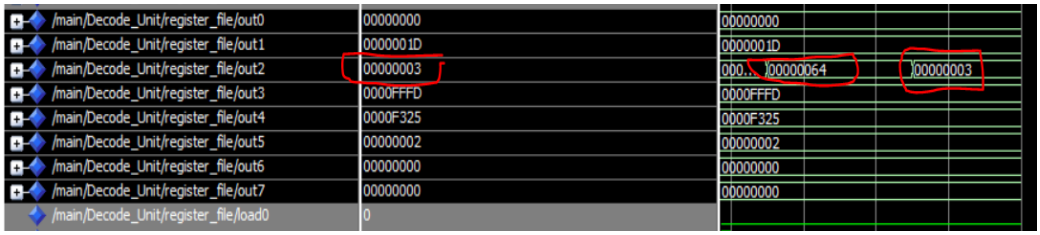
As shown in (Fig. 11) R5 is 00000002 not FFFF, this happened because that instruction 5 depends on instruction 3 which has not written back in R3 yet.

- Read after write hazard detected between instruction 7 and instruction 5 and 6. See (Fig. 12)



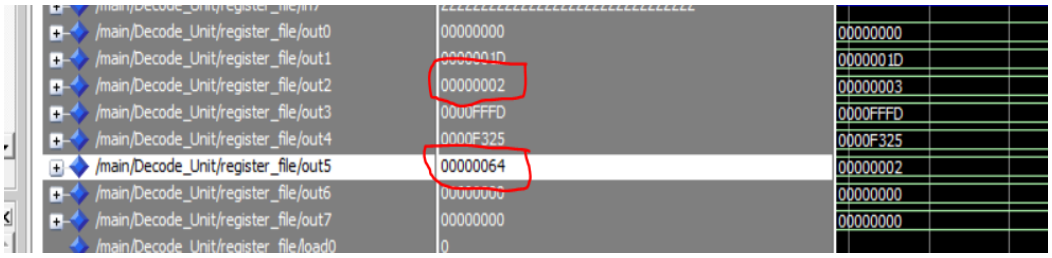
(Fig. 12)

- Read after write hazard detected between instruction 7 and instruction 8.
- Read after write hazard detected between instruction 10 and instruction 11. See (Fig. 13)



(Fig. 13)

- Read after write hazard detected between instruction 12 and instruction 11. See (Fig. 14)



(Fig. 14)

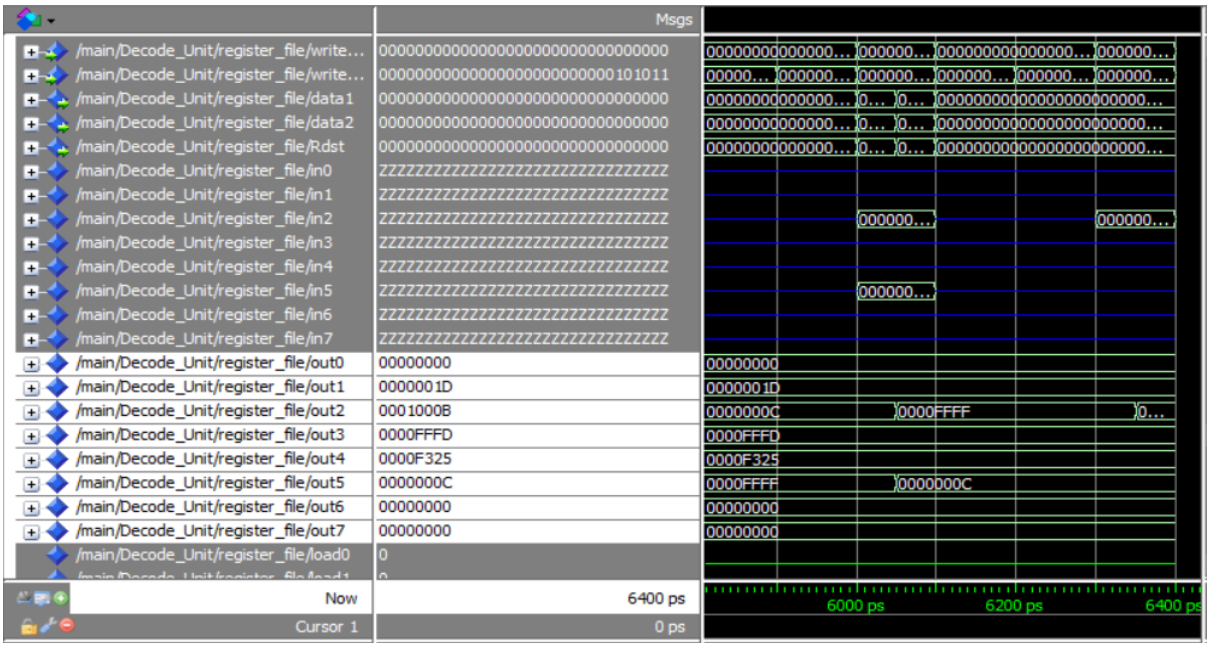
- Read after write hazard detected between instruction 12 and instruction 13

- **Solving the hazards by inserting no operations**

```
1  in   R1       #add 5 in R1
2  in   R2       #add 19 in R2
3  in   R3       #FFFD
4  in   R4       #F320
5  nop
6  IADD R3,R5,2  #R5 = FFFF , flags no change
7  ADD  R1,R4,R4    #R4= F325 , C-->0, N-->0, Z-->0
8  nop
9  nop
10 SUB  R5,R4,R6    #R6= 0CDA , C-->1, N-->0,Z-->0
11 nop
12 nop
13 AND  R7,R6,R6    #R6= 00000000 , C-->no change, N-->0, Z-->1
14 OR   R2,R1,R1    #R1=1D  , C--> no change, N-->0, Z--> 0
15 SHL  R2,2     #R2=64  , C--> 0, N -->0 , Z -->0
16 nop
17 nop
18 SHR  R2,3     #R2=0C  , C -->1, N-->0 , Z-->0
19 nop
20 nop
21 SWAP R2,R5    #R5=0C ,R2=FFFF  ,no change for flags
22 nop
23 nop
24 ADD  R5,R2,R2    #R2= 1000B (C,N,Z= 0)
```

(Fig. 15)

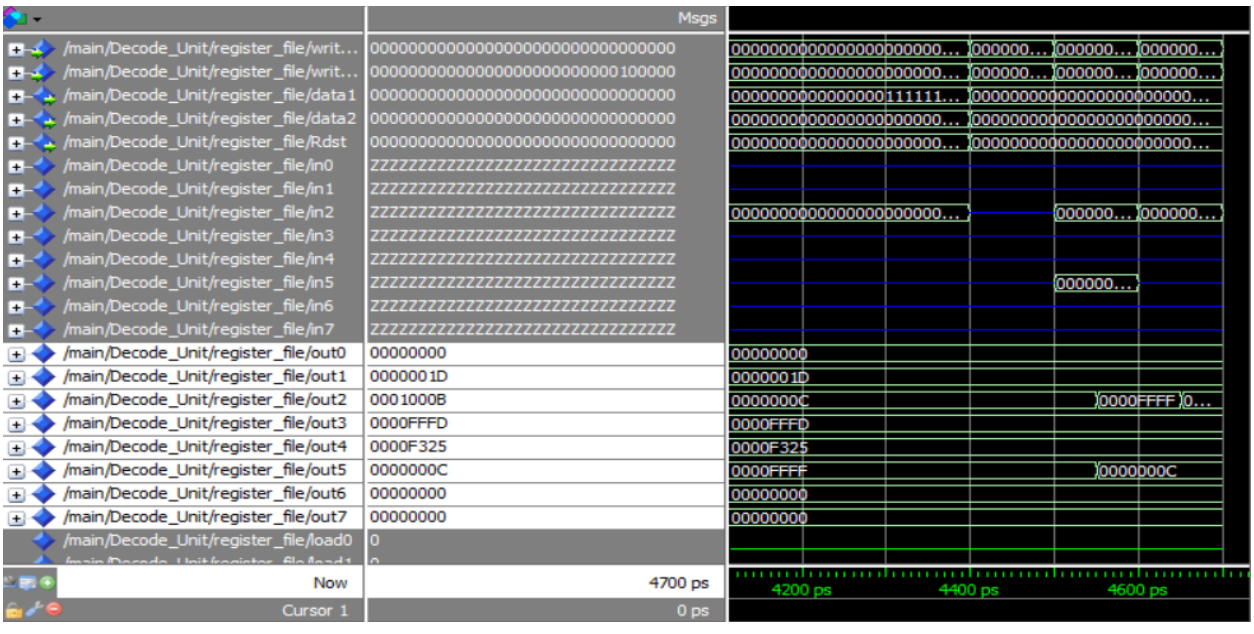
As shown in (Fig. 15), we inserted no operations between any dependencies to prevent hazards. The output contains the right values after these modifications (look Fig. 16).



(Fig. 16)

2. With the forwarding units added

By adding the forwarding units there is no hazards and the output was correct (look Fig. 17)



(Fig. 17)

3. Speed up due to adding the forwarding units.

As shown in (Fig. 16), in case of no forwarding units and handling hazards by inserting no operations, the program is executed in 6400 ps.

As shown in (Fig. 17), after adding all forwarding units, the program is executed in 4600 ps.

So the speed up = 6400/4600 = 1.4.

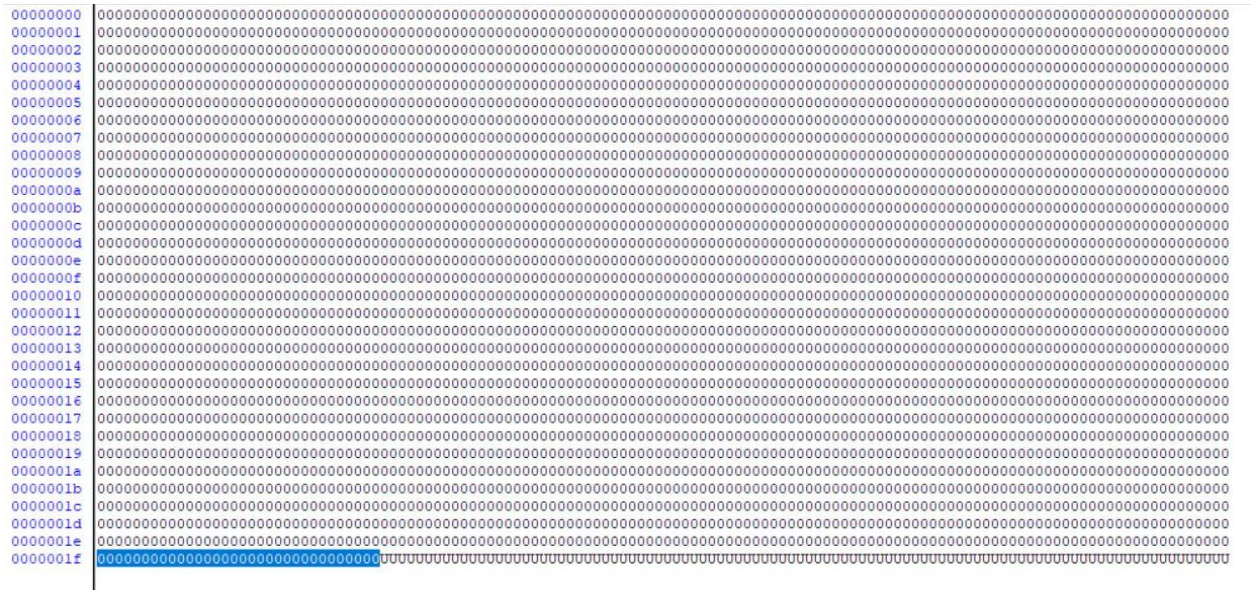
Memory

Note ram is filled with garbage (UUUUU) in the beginning.
Assuming cache is 1 indexed.

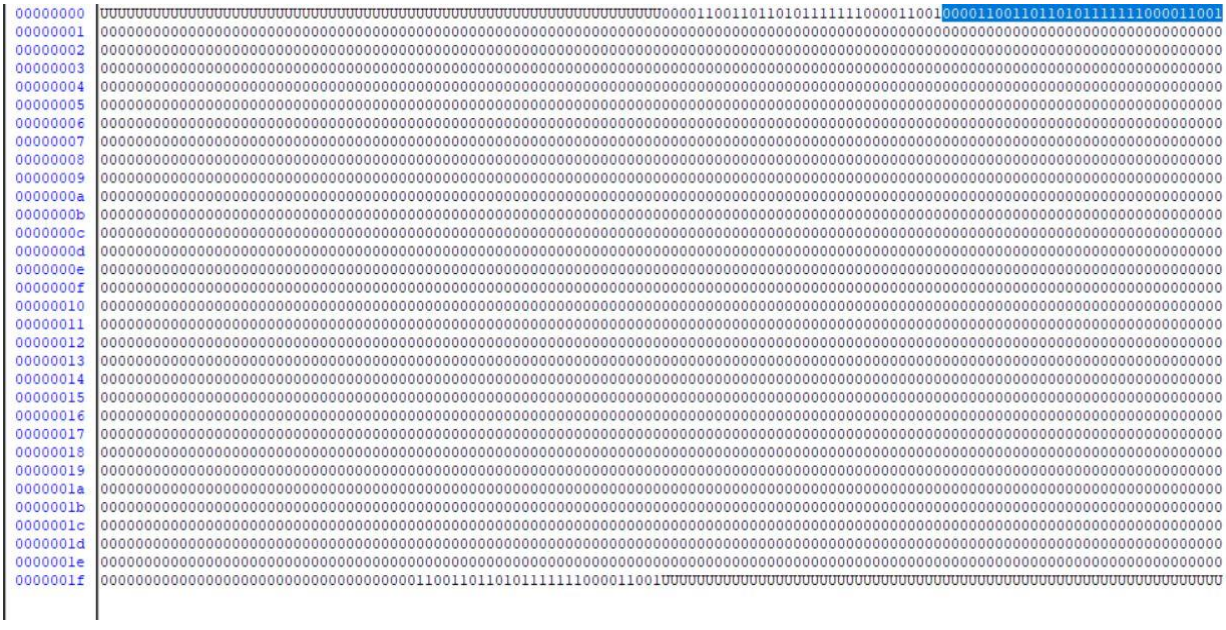
Code

```
.ORG 2  #this is the interrupt address
100

.ORG 10
in R2          #R2=0CDAFE19 add 0CDAFE19 in R2
in R3          #R3=FFFF
in R4          #R4=F320
LDM R1,F5      #R1=F5
PUSH R1        #SP=7FC, M[7FE, 7FF] = F5
PUSH R2        #SP=7FA,M[7FC, 7FD]=0CDAFE19
POP R1         #SP=7FC,R1=0CDAFE19
POP R2         #SP=7FE,R2=F5
STD R2,200     #M[200, 201]=F5
STD R1,202     #M[202, 203]=0CDAFE19
LDD R3,202     #R3=0CDAFE19
LDD R4,200     #R4=5
```

1. Without Forward unit and hazard detection
    - Read after write in PUSH R1
        - cache[32][4] = F5 cache[32][4] = 0



    - Read after write in STD R2,200
        - cache[0][0] = 0 depending on wrong push cache[0][0] = 0CDAFE19



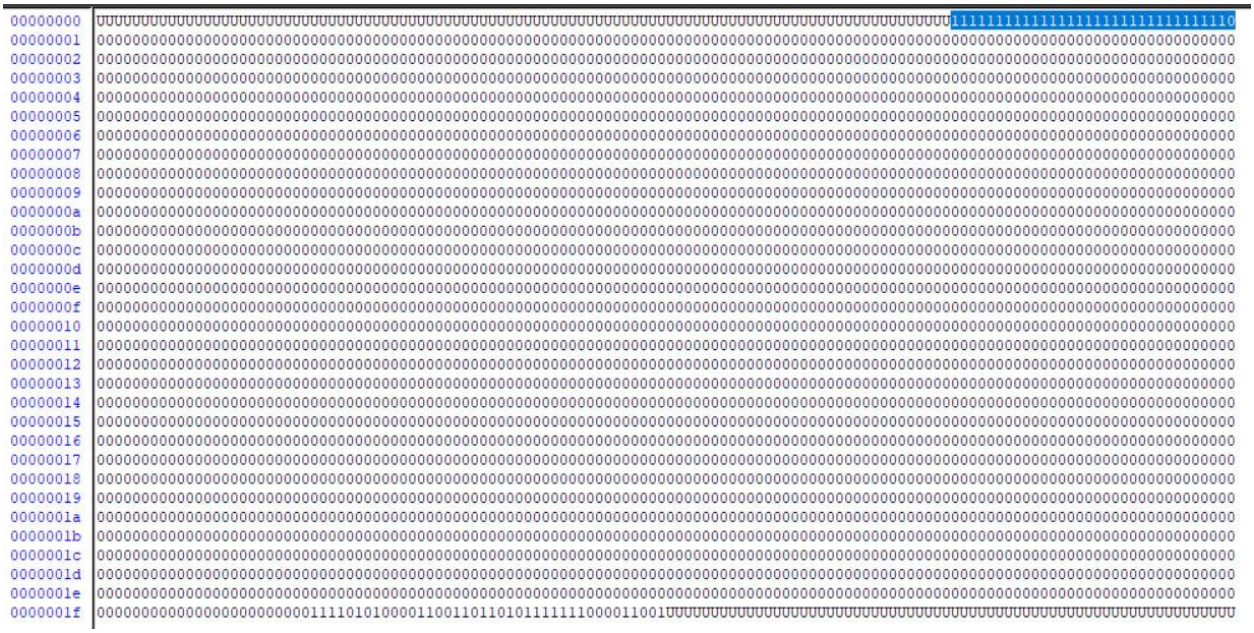
Solution : add **2** no operations before **each** hazard.

2. With Forward unit only

- Read after write in STD R2,200

  Forward unit forwards wrong data (sp) as no hazard detection unit to stall processor [load use case]

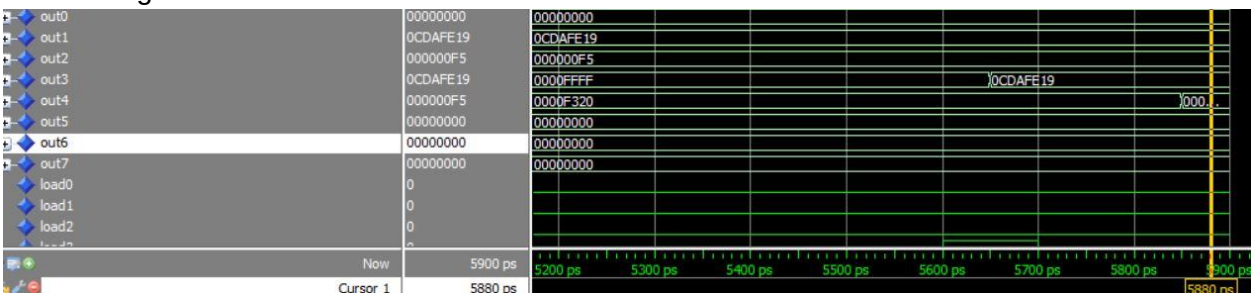
  - cache[0][0] = F5 cache[0][0] = FFFFFFFE



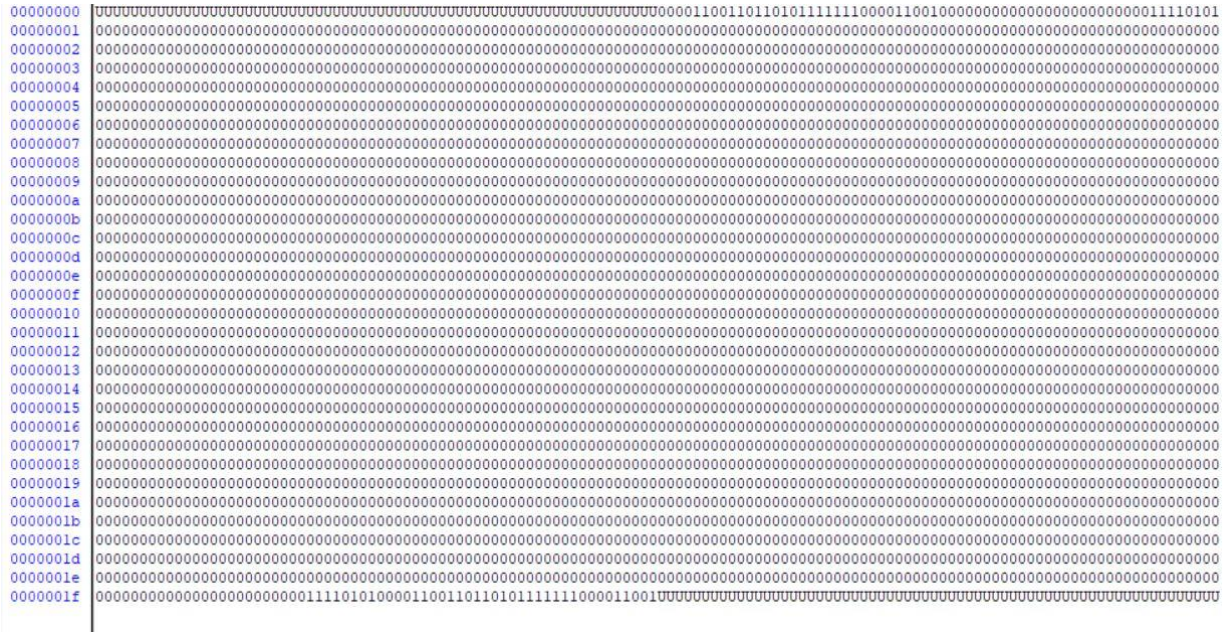
- Solution to add **2** no operations before STD R2,200

3. With Forward unit and hazard detection

- Registers values are valid



- Memory data is valid



Hazard detection units are important for validating data.
The forward unit overcomes two installs which will also increase 4 cycles to fetch new block from ram to instruction cache.
Speed up = (6500/5900) = 1.1

Cache

Substitute call R0 → jmp r0.

Ret → LDM R7, 1B

JMP R7

So returning 3B is impossible so the test case is valid until 18850ps.

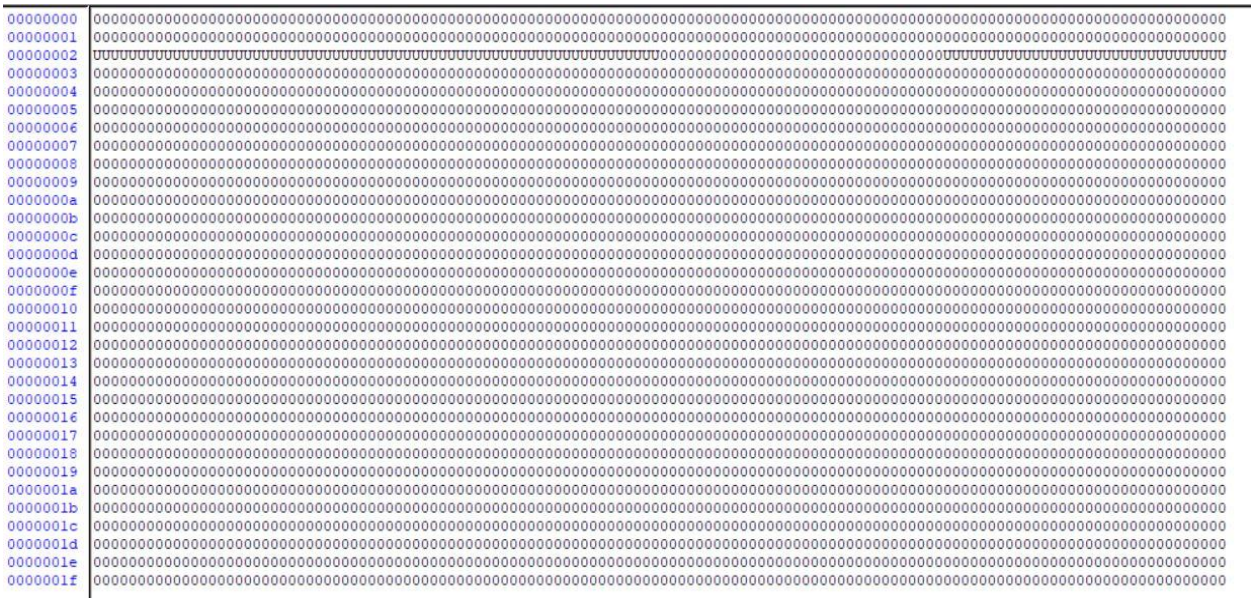
1. Without Forward unit and hazard detection

- Read after write line 49 : STD R6,312
  - cache[3][2] = 0 cache[3][2] = 0

  Lucky wrong data is the same as valid data

  Error is repeated each iteration

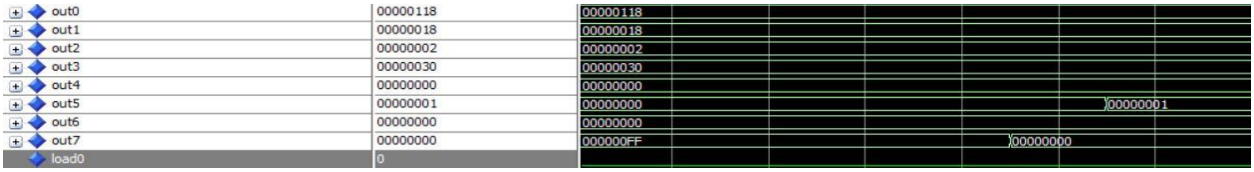
  Solution : add 2 no operations before this instruction



- Read after write line 53 : OR R5,R5,R6
  - R[6] = 1 R[6] = 0

  Error is repeated each iteration

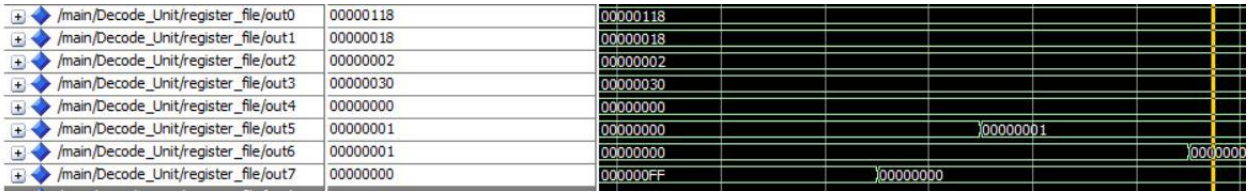
  Solution : add 2 no operations before this instruction



Having no flushes will execute two instructions after each jump or call.

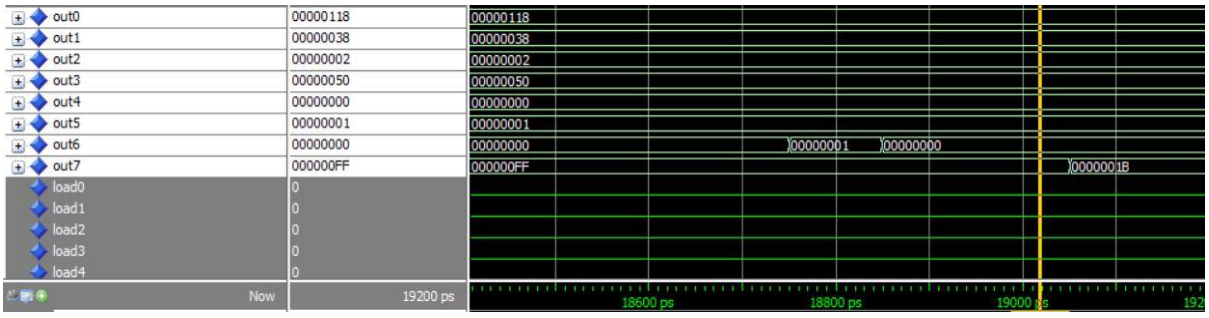
2. With forward unit

- All problems are solved
  R[6] = 1 after OR R5,R5,R6



Adding a Hazard detection unit won't change anything as forward units solves all problems.

Using forward units the test case runs at 18850ps.



Inserting stalls the case runs at 20600ps.
Speed up = (20600/18850) = 1.09

Branch

Unhandled instructions

- CALL instruction: replaced with JMP.
- RET instruction: replaced with LDM with any not used reg and JMP.

Modified code

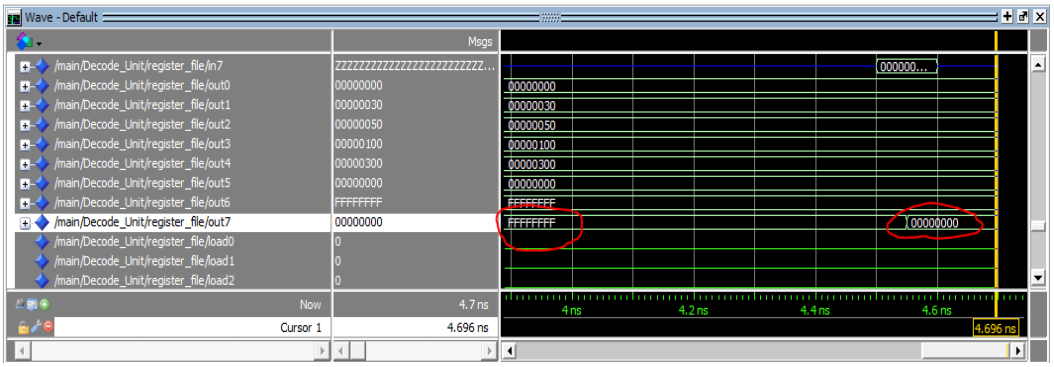
```
.ORG 0  #this means the the following line would be  at address  0 , and this
is the reset address
10
#you should ignore empty lines
.ORG 2  #this is the interrupt address
100
.ORG 10
in R1      #R1=30
in R2      #R2=50
in R3      #R3=100
in R4      #R4=300
in R6      #R6=FFFFFFFF
in R7      #R7=FFFFFFFF
Push R4    #sp=7FC, M[7FE, 7FF]=300
JMP R1
INC R7     # this statement shouldn't be executed,
#check flag forwarding
.ORG 30
AND R1,R5,R5    #R5=0 , Z = 1
            #try interrupt here
JZ  R2      #Jump taken, Z = 0
INC R7      #this statement shouldn't be executed
#check on flag updated on jump
.ORG 50
JZ R3       #Jump Not taken
#check destination forwarding
NOT R5      #R5=FFFFFFFF, Z= 0, C--> not change, N=1
INC R5      #R5=0, Z=1, C=1, N=0
in  R6      #R6=200, flag no change
JZ  R6      #jump taken, Z = 0
INC R1    #this statement shouldn't be executed
.ORG 100
ADD R0,R0,R0     #N=0,Z=1,C=0
out R6
rti
#check on load use
.ORG 200
POP R6      #R6=300, SP=7FE
```

```
JMP R6     #SP=7FC, M[7FF]=half next PC,M[7FE]=other half next PC
           #try interrupt here
INC R6     #R6=401, this statement shouldn't be executed till call returns, C-
-> 0, N-->0,Z-->0
NOP
NOP
.ORG 300
Add R3,R6,R6 #R6=400
Add R1,R2,R1 #R1=80, C->0,N=0, Z=0
LDM R0, 202
JMP R0
INC R7            #this should not be executed
.ORG 500
NOP
NOP
```

1. Without Forward unit and hazard detection

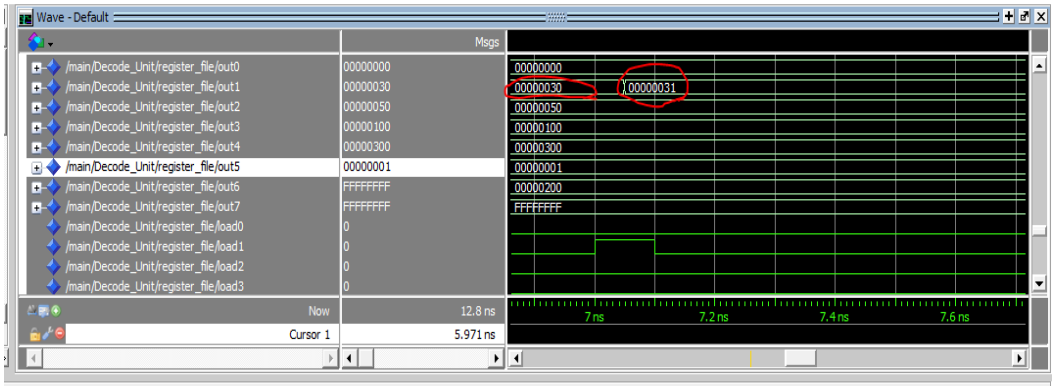
- **Hazards types:**
  - Control hazards as in lines 20 and 21:

```
JMP R1
INC R7     # this statement shouldn't be executed.
```



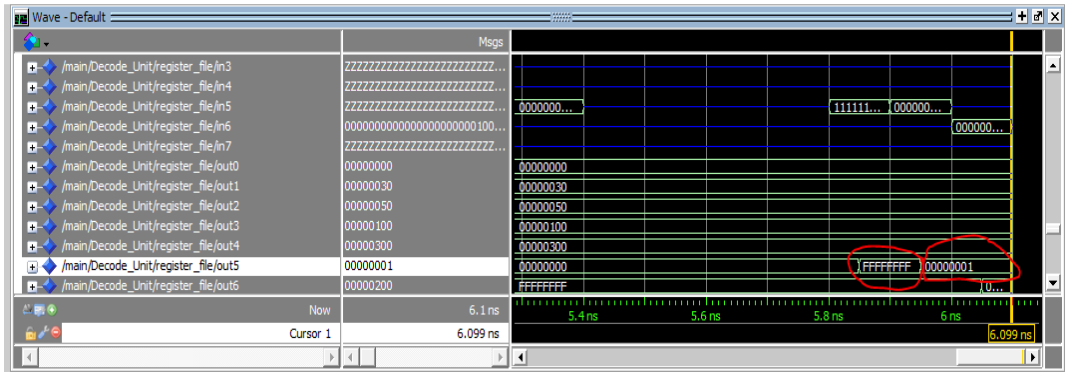
As shown in the figure: the increment instruction is executed.
Similarly, for the instructions in lines 38 and 39.

```
JZ   R6      #jump taken, Z = 0
INC  R1    #this statement shouldn't be executed
```



- Data hazards:
  1. Read after write for R5 in lines 35 and 36:

```
NOT R5      #R5=FFFFFFFF, Z= 0, C--> not change, N=1
INC R5      #R5=0, Z=1, C=1, N=0
```

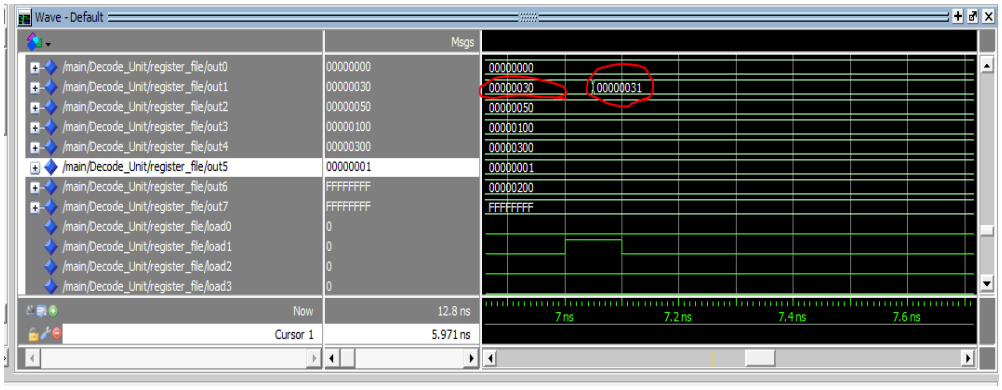


As shown in the figure the increment instruction is executed on the old value of R5 which is zero instead of using the new value resulted after not instruction.

2. Read after write for R6 in lines 37 and 38:

```
NOT R5      #R5=FFFFFFFF, Z= 0, C--> not change, N=1
INC R5      #R5=0, Z=1, C=1, N=0
in  R6      #R6=200, flag no change
JZ  R6      #jump taken, Z = 0
INC R1      #this statement shouldn't be executed
```

Another problem is that Zero flag is lost due to data hazard in R5. So, the jump will not be executed and the increment instruction will be executed as shown in the figures.



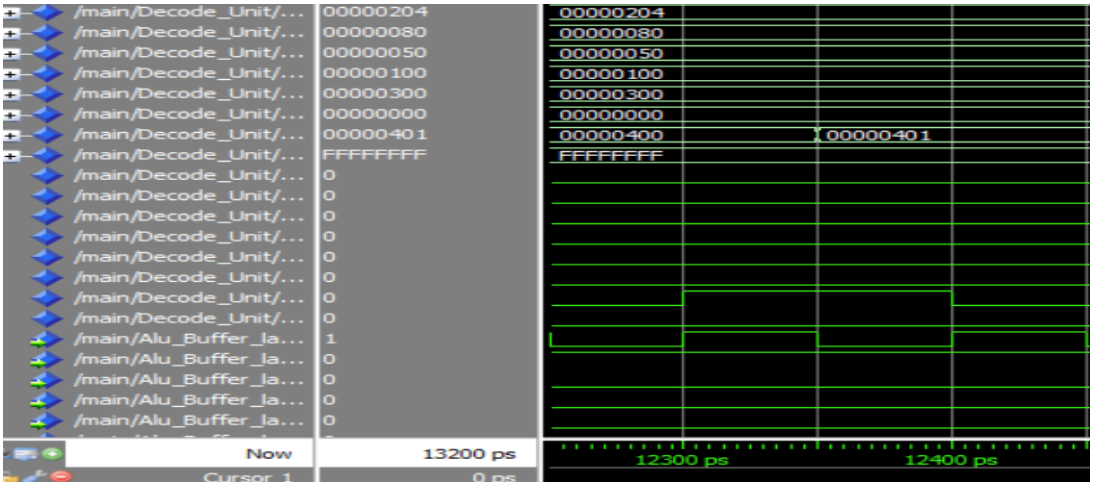
3. Load use case for R6 in lines 48 and 49:

```
POP R6      #R6=300, SP=7FE
JMP R6     #SP=7FC, M[7FF]=half next PC,M[7FE]=other
half next PC
```

- **Solving hazards using no operations the processor ends in 12350 ps with following code:**

```
.ORG 0  #this means the the following line would be  at address
0 , and this is the reset address
10
#you should ignore empty lines

.ORG 2  #this is the interrupt address
100

.ORG 10
in R1      #R1=30
in R2      #R2=50
in R3      #R3=100
in R4      #R4=300
in R6      #R6=FFFFFFFF
in R7      #R7=FFFFFFFF
nop
Push R4    #sp=7FC, M[7FE, 7FF]=300
JMP R1
INC R7     # this statement shouldn't be executed,

#check flag forwarding
.ORG 30
AND R1,R5,R5   #R5=0 , Z = 1
JZ  R2       #Jump taken, Z = 0
INC R7       #this statement shouldn't be executed

#check on flag updated on jump
.ORG 50
JZ R3       #Jump Not taken

#check destination forwarding
NOT R5      #R5=FFFFFFFF, Z= 0, C--> not change, N=1
Nop
Nop
INC R5      #R5=0, Z=1, C=1, N=0
in  R6      #R6=200, flag no change
nop
nop
JZ  R6      #jump taken, Z = 0
INC R1    #this statement shouldn't be executed

.ORG 100
ADD R0,R0,R0     #N=0,Z=1,C=0
out R6
rti

#check on load use
```

```
.ORG 200
POP R6      #R6=300, SP=7FE
nop
nop
jmp R6     #SP=7FC, M[7FF]=half next PC,M[7FE]=other half next PC
nop
nop
INC R6     #R6=401, this statement shouldn't be executed till call
returns, C--> 0, N-->0,Z-->0
NOP
NOP


.ORG 300
Add R3,R6,R6 #R6=400
Add R1,R2,R1 #R1=80, C->0,N=0, Z=0
LDM R0, 204
nop
nop
jmp R0
nop
nop
INC R7            #this should not be executed

.ORG 500
NOP
NOP
```



2. With forwarding unit

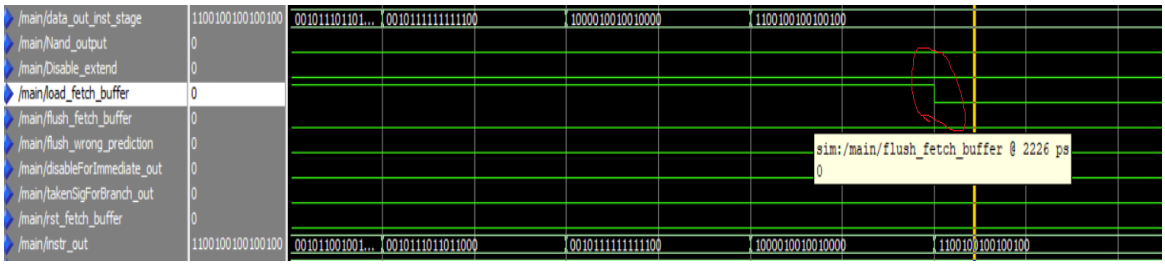
- **Hazards types:**
  - Control hazards as in lines 20 and 21:

```
JMP R1
INC R7     # this statement shouldn't be executed.
```

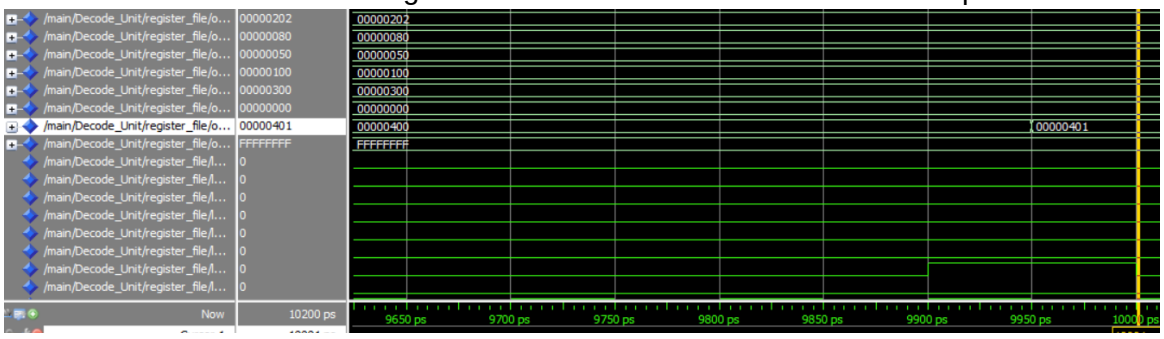
Similarly, for the instructions in lines 38 and 39.

```
JZ  R6      #jump taken, Z = 0
INC R1    #this statement shouldn't be executed
```

- **Solving hazards by stalling fetch when a branch operation is in the fetch stage until jmp is executed and Pc is updated :**



This is result with full forwarding and hazards removal which runs in 9950 ps



3. Calculating Speed up on using both forwarding and hazard detection units

Speed up = (12350/9950)=1.24

Not working modules

- Branch prediction is not working correctly.it is fully implemented but there is a bug and there is no time to fix it.