Hazard Logic

1) **Forwarding Unit:**

This hazard is because of the Data Dependency
Like rf_a1_RR = rf_a3_EX or rf_a2_RR = rf_a3_EX , Register wont be updated by the time next instruction needs it
So Forwarding unit forwards the required Register information observing the above like registers

We divide the Forwarding Unit into 3 cases

Case 1
rf_a3_RR/EX/WB = rf_a1_RR

if (rf_a1_RR = rf_a3_EX) and (wr_en_EX = '1') and not(main_opcode_RR = "**SW**" or main_opcode_RR = "**LW**") and flush_EX_t = '0' then
  FU_out1 <= alu1_out; **— LW We have to store in RA (so no data dependency)**
  — SW we defined in Case 3

elsif (main_opcode_RR = "0111" or main_opcode_RR = "0110") and pc_RR = pc_EX and flush_EX_t = '0' then **--LM/SM data forwarding**
  FU_out1 <= alu1_out; **— Computing address for next unit we access previous and add 1 till the LM/SM is executed fully**

elsif (rf_a1_RR = rf_a3_MEM) and (wr_en_MEM = '1') and not(main_opcode_RR = "**SW**" or main_opcode_RR = "**LW**") and flush_MEM = '0' then
  FU_out1<= mem_out; **— LW We have to store in RA (so no data dependency)**
  — SW we defined in Case 3

elsif (rf_a1_RR = rf_a3_WB) and (wr_en_WB = '1') and not(main_opcode_RR = "**LW**" or main_opcode_RR = "**SW**") and flush_WB = '0' then
  FU_out1 <= wb_output; **— LW We have to store in RA (so no data dependency)**
  — SW we defined in Case 3

An important point to note is we only consider when **wr_en_EX/MEM/WB = 1**
If **wr_en_EX/MEM/WB = 0** means not writing back into registers implies
no Data Dependency

Case 2
rf_a3_RR/EX/WB = rf_a2_RR

if (rf_a2_RR = rf_a3_EX) and (wr_en_EX = '1') and not(main_opcode_RR = "**LM**" or main_opcode_RR = "**ADI**" or main_opcode_RR = "**SM**" or main_opcode_RR = "**LHI**") and flush_EX_t = '0' then
  FU_out2 <= alu1_out;

elsif (rf_a2_RR = rf_a3_MEM) and (wr_en_MEM = '1') and not(main_opcode_RR = "**LM**" or main_opcode_RR = "**SM**" or main_opcode_RR = "**ADI**" or main_opcode_RR = "**LHI**") and flush_MEM = '0' then
  FU_out2<= mem_out;

elsif (rf_a2_RR = rf_a3_WB) and (wr_en_WB = '1') and not(main_opcode_RR = "**LM**" or main_opcode_RR = "**SM**" or main_opcode_RR = "**ADI**" or main_opcode_RR = "**LHI**") and flush_WB = '0' then
  FU_out2 <= wb_output;

— there is no RB for LM/SM/ADI/LHI
For all other cases, it is trivial

Case 3
For SM/SW case

if (main_opcode_RR = "**SW**" or main_opcode_RR = "**SM**") then

if ((main_opcode_EX = "**LM**" or main_opcode_EX = "**LW**") and (rf_a1_RR = rf_a3_EX) and (wr_en_EX = '1') and flush_EX_t = '0') then
        FU_temp <= rf_d1/rf_d2;   **rf_d1 for SW & rf_d2 for SM**
        sw_data_forward_RR <= '1';

In all other cases except for LOAD followed STORE, we generally STALL the stages
Here for STORE, we won't we computing in ALU and we need to have data by the time we access Memory Block so we get this data in EX STAGE instead of RR STAGE (which needs Stalling)
For that, we assign a register sw_data_forward, we then access MEM_out in EX stage if sw_data_forward is set (which implies previous instruction was LOAD)

```
elsif (rf_a1_RR = rf_a3_EX) and (wr_en_EX = '1') and flush_EX_t = '0' then
        FU_temp <= alu1_out;
        sw_data_forward_RR <= '0';

elsif (rf_a1_RR = rf_a3_MEM) and (wr_en_MEM = '1') and flush_MEM = '0' then
        FU_temp<= mem_out;
        sw_data_forward_RR <= '0';

elsif (rf_a1_RR = rf_a3_WB) and (wr_en_WB = '1') and flush_WB = '0' then
        FU_temp <= wb_output;
        sw_data_forward_RR <= '0';
```

The above 3 are trivial!

2) **LM SM stalling**:

We need to stall the IF stage when the instruction is at ID stage
As we need to load Multiple Registers which takes multiple cycles
(because we need to check for **pe_zero** which decides the stalling and that is in ID_RR pipe)

stall_IF <= '1' when (((pe_zero = '0' and (imem_ID(15 downto 12) = "**LM**" or i_mem_ID(15 downto 12) = "**SM**")) and flush_ID_t = '0')

3) **Load Immediate Dependency:** Detected in the EX stage, but the Data is available from the MEM Stage. Hence, we need one stall for the instruction in the IF, ID and RR stages

```
If (rf_a3_ex = rf_a1_rr or rf_a3_ex = rf_a2_rr) and ( opcode_EX = “LOAD”) then
IF_ID_en <= ‘0’;
ID_RR_en <= ‘0’;
RR_EX_en <= ‘0’;
Pc_en <= ‘0’;
I_mem_rd <= ‘0’;
Else
IF_ID_en <= ‘1’;
ID_EX_en <= ‘1’;
EX_RR_en <= ‘1’;
Pc_en <= ‘1’;
I_mem_rd <= ‘1’;
If (rf_a2_rr = rf_a3_ex) and (wr_en_ex = ‘1’) then
reg_in2 <= alu_out;
```

```
Elsif (rf_a2_rr = rf_a3_mem) and (wr_en_mem = ‘1’) then
reg_in2<= mem_out;
Elsif (rf_a2_rr = rf_a3_wb) and (wr_en_wb = ‘1’) then
reg_int2 <= wb_out;
Elsif()
reg_in2<=”0000000000000001”;
reg_in2 <= rf_d2
End if;
```

Summing up above **LM/SM and Load Dependency case** following we introduce

4) **STALLING BLOCK:**

We divide STALL into 3 cases

Case 1

When is pe_zero is not set and the LM/SM is in the ID stage

```
if(pe_zero = '0' and (imem_ID(15 downto 12) = "LM" or imem_ID(15 downto 12) = "SM")) and
flush_ID_t = '0' then
if ((rf_a3_EX = rf_a1_RR or rf_a3_EX = rf_a2_RR) and (main_opcode_EX = "LOAD")) and
flush_EX_t = '0'
stall_IF <= '1';
stall_ID <= '1';          --- Basic LOAD ( LOAD Dependency )
stall_RR <= '0';

elsif rf_a3_EX = rf_a2_RR and main_opcode_EX = "LM" and not (main_opcode_RR = "LM" or
main_opcode_RR = "SM" or main_opcode_RR = "ADI" or main_opcode_RR = "LHI") and flush_EX_t
= '0' then
stall_IF <= '1';          — opcode in next stage(EX) = “LM”
stall_ID <= '1';            -- for ADI/LHI RB is used for storing result (no dependency)
stall_RR <= '0';          -- for LM/SM there is no RB

elsif rf_a3_EX = rf_a1_RR and main_opcode_EX = "LM" and not (main_opcode_RR = "LW" or
main_opcode_RR = "SW") and flush_EX_t = '0' then
          stall_IF <= '1';
  stall_ID <= '1';        — LM we have to store in address RA
  stall_RR <= '0'         — SM, we will set sw_forward_data and get data in EX stage
                            Instead of stalling a stage
```

```
else
  stall_IF <= '1';        — Basic LM/SM stalling
  stall_ID <= '0';
  stall_RR <= '0';
end if;
```

Case 2

When is pe_zero is set and the LM/SM is in the ID stage

```
elsif (pe_zero = '1' and (imem_ID(15 downto 12) = "0110" or imem_ID(15 downto 12) =
"0111")) and flush_ID_t = '0' then

if ((rf_a3_EX = rf_a1_RR or rf_a3_EX = rf_a2_RR) and (main_opcode_EX = "LOAD")) and
flush_EX_t = '0' then
  stall_IF <= '1';          --- Basic LOAD ( LOAD Dependency )
          stall_ID <= '1';
          stall_RR <= '0';

elsif((rf_a3_EX = rf_a1_RR or rf_a3_EX = rf_a2_RR) and (main_opcode_EX = "LM"))   and
flush_EX_t = '0' then
          stall_IF <= '1';          — opcode in next stage(EX) = “LM”
          stall_ID <= '1';          — Its the same as Basic LOAD dependency
          stall_RR <= '0';
  else
          stall_IF <= '0';
          stall_ID <= '0';
          stall_RR <= '0';
```

Case 3

Only LOAD Dependency

```
elsif ((rf_a3_EX = rf_a1_RR or rf_a3_EX = rf_a2_RR) and (main_opcode_EX = "LW")) and
flush_EX_t = '0' then --opcode in next stage(EX) = “LOAD”
  stall_IF <= '1';
  stall_ID <= '1';
  stall_RR <= '0';
```

Case 4
If not of LW/SW or LOAD then no stalling

```
else
  stall_IF <= '0';
  stall_ID <= '0';
  stall_RR <= '0';
```

5) **FLUSH BLOCK**

Case 1
This block is used to delete the Instructions in the ID, IF, RR when the **Branch is taken**
We by default consider the **Branch is not taken,** so if the Branch is taken we have to delete the next 3 instruction because they are
Case 2
Also in case of Jump, we get PC in **RR stage,** so we have to Flush/delete next 2 instructions and update the **PC**

Case 1

```
if alu1_in1 = alu1_in2 and main_opcode_EX = "BEQ" and flush_EX_t = '0' then
  flush_IF_t <= '1';
  flush_ID_t <= '1';                          — BEQ
  flush_RR_t <= '1';
  flush_EX_t <= flush_EX;
```

Case 2

```
elsif (main_opcode_RR ="JLR" or main_opcode_RR = "JAL") and flush_RR_t ='0' then
  flush_IF_t <= '1';
  flush_ID_t <= '1';                          — JAL/JLR
  flush_RR_t <= flush_RR;
  flush_EX_t <= flush_EX;
```

Case 3

```
elsif (rf_a3_EX = rf_a1_RR or rf_a3_EX = rf_a2_RR) and (main_opcode_EX = "LW") and flush_EX_t
= '0' then
  flush_IF_t <= flush_IF;                -- Basic LOAD ( LOAD Dependency )
  flush_ID_t <= flush_ID;
flush_RR_t <= '1';
flush_EX_t <= flush_EX;
```

— Here we are stalling the following instructions, then there is nothing to be executed in the Execution Stage in the following instruction so we just Flush it

The following 2 are the same cases we used for stalling and above Reasoning works

```
elsif (rf_a3_EX = rf_a1_RR)  and (main_opcode_EX = "LM" and not (main_opcode_RR = "LW" or
main_opcode_RR = "SW")) and flush_EX_t = '0' then
  flush_IF_t <= flush_IF;
  flush_ID_t <= flush_ID;   — opcode in next stage(EX) = “LM”
  flush_RR_t <= '1';        — for ADI/LHI RB is used for storing result (no dependency)
  flush_EX_t <= flush_EX;  -- for LM/SM there is no RB
```

```
elsif (rf_a3_EX = rf_a2_RR)  and (main_opcode_EX = "LM" and not (main_opcode_RR = "LM" or
main_opcode_RR = "SM" or main_opcode_RR = "ADI" or main_opcode_RR = "LHI")) and
flush_EX_t = '0' then
  flush_IF_t <= flush_IF;     — LM we have to store in address RA
  flush_ID_t <= flush_ID;     — SM, we will set sw_forward_data and get data in EX stage
  flush_RR_t <= '1';            Instead of stalling a stage
  flush_EX_t <= flush_EX;
```

Following are the cases when the **destination register is R7 (which is assigned for storing PC)**

```
elsif (rf_a3_EX = "111" and wr_en_EX = '1') and not (main_opcode_EX = "LW" or main_opcode_EX
= "LM") and flush_EX_t ='0' then
  flush_IF_t <= '1';
  flush_ID_t <= '1';          — We update PC directly from alu_output and Flush
  flush_RR_t <= '1';            all 3 following instructions
  flush_EX_t <= flush_EX;  — For LM/LW alu output will give the address of Memory which
                                should be loaded into R7 which will be available in MEM stage
```

```
elsif (rf_a3_MEM = "111" and wr_en_MEM = '1') and (main_opcode_MEM = "LW" or
main_opcode_MEM = "LM") and flush_MEM ='0' then      --for Load and LM--4 cycles are to be
flushed
  flush_IF_t <= '1';      — as mentioned above, for LM/LW we access output
  flush_ID_t <= '1';       from MEM stage to update PC then flush the following 4 instructions
  flush_RR_t <= '1';
  flush_EX_t <= '1';


If no hazard no worries

else
  flush_IF_t <= flush_IF;
  flush_ID_t <= flush_ID;
  flush_RR_t <= flush_RR;
  flush_EX_t <= flush_EX;
end if;
```

6) ADC after ADD or something

For this, we define Carry/Zero Flags at the WB Stage
A block checks if the output is zero or not
Carry flag along with Zero flag, gets updated based on the previous Carry_flag / Zero_flag (being set or not) according to the present Instruction (ADC/ADZ)