HAZARD HANDLING

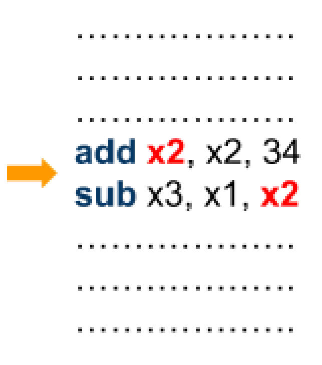
There are three major types of hazards.

1. Data Hazards
2. Control Hazards
3. Structure Hazards

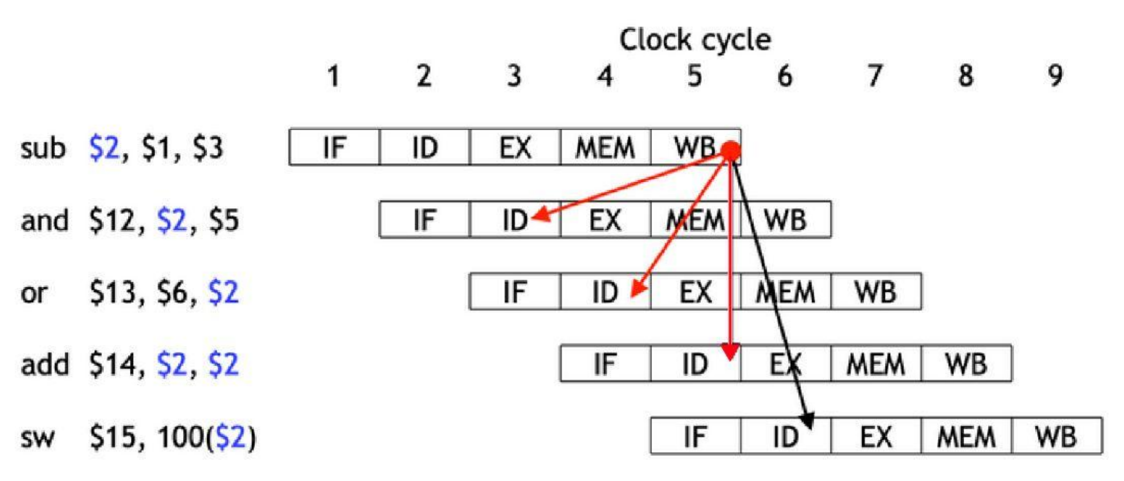
In our five-stage pipeline design, there are no structure hazards because we have implemented in order execution architecture. So, there is no resource sharing happening in our pipeline.

- ***Data Hazard Handling***

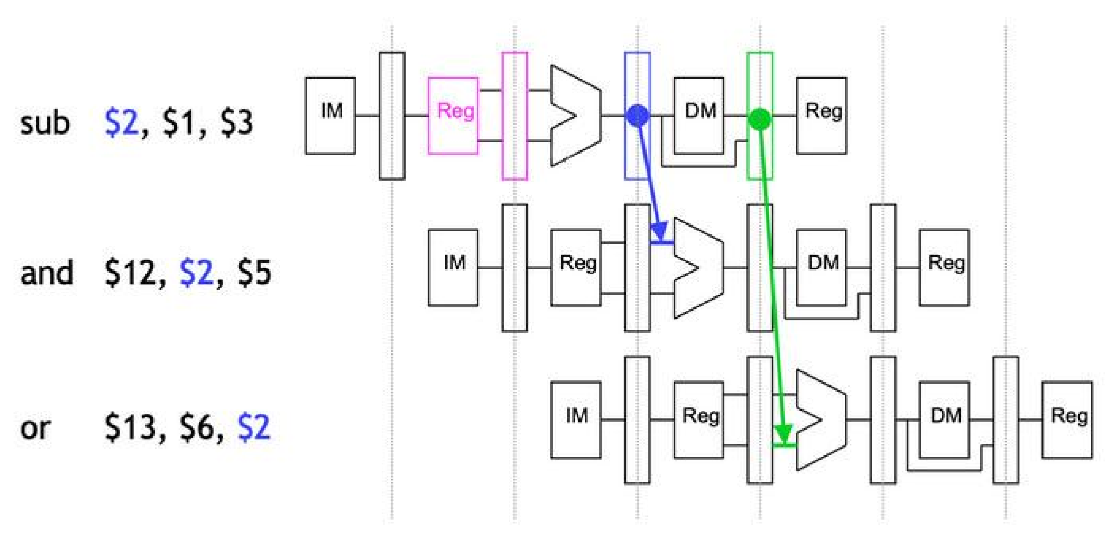
Data hazards mean the data dependencies between instructions. We have implemented two forwarding units inside stage three and stage 4. There are several methods to handle data hazards. The easy and basic method is to insert bubbles into the pipeline if there are data dependencies. This is an easy approach, but this method decreases the efficiency of the pipeline. The second method is to use forwarding methods. That means taking the ALU results from stages 4 and 5 as operand 1 and operand 2 in the execution stage. The following diagram shows the results that should be forwarded in order to handle the dependency data hazards.



```
………………
………………
………………
add x2, x2, 34
sub x3, x1, x2
………………
………………
………………
```



The following diagram shows how we handle the forwarding in various stages.



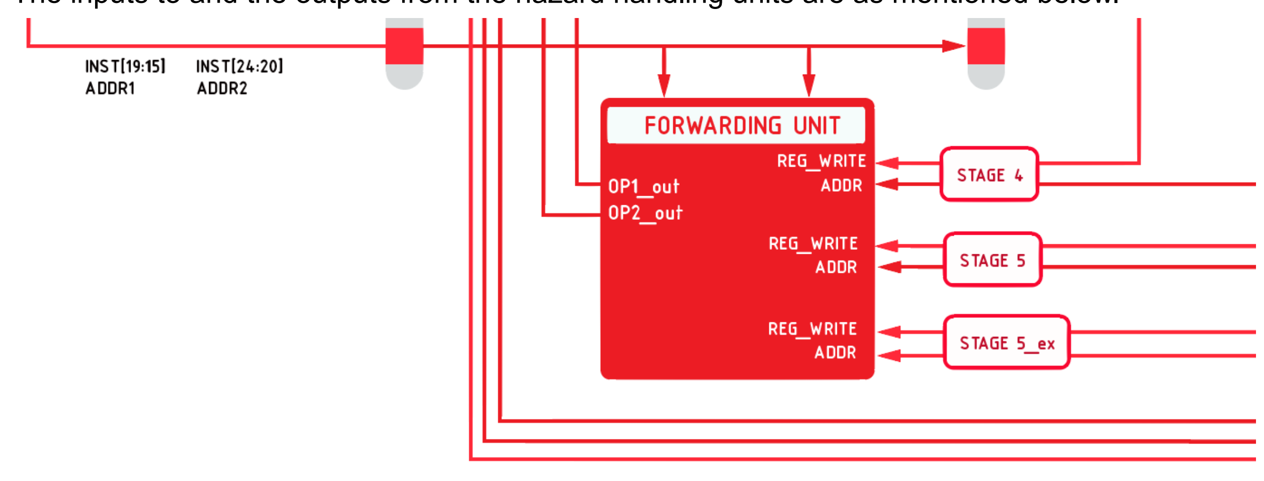
The following diagram shows the data and control path with the whole CPU with forwarding units.

Stage 3 (Execution Stage) forwarding unit

This forwarding unit keeps the eye on operand 1 and operand 2 and checks whether there are new values for those lines in Stage 4 or Stage 5. If that is the case the forwarding unit sends the signal to the Hazard Handling Muxes in operand 1 and operand 2 data paths.

Data dependency could happen even between 3 stages apart. To be more clear, data that are in the write back stage are not written to the reg file while the instruction at the decode stage fetches old data from the reg file. Therefore the forwarding unit should forward the data that is currently in the writeback stage at the next clock edge. Since there is no buffer to hold the data that will be omitted from the stage 5 (WB) pipeline register. To hold those data an additional pipeline register set was used.

The inputs to and the outputs from the hazard handling units are as mentioned below.



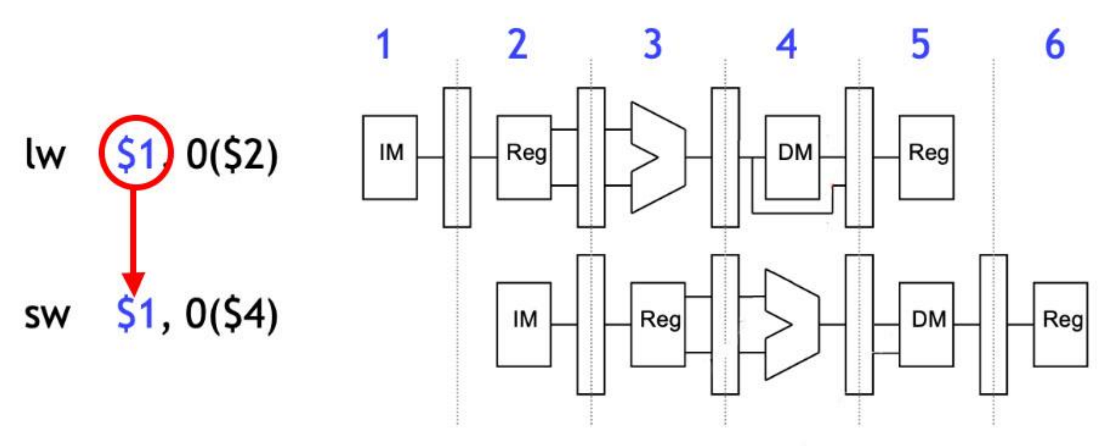
operand 1 forwarding MUX	
00	Regular Value
01	Forwarded Stage 4
10	Forwarded Stage 5
11	Forwarded extra stage

operand 2 forwarding MUX	
00	Regular Value
01	Forwarded Stage 4
10	Forwarded Stage 5
11	Forwarded extra stage

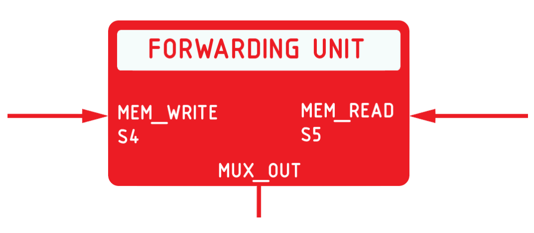
Stage 4 (Main Memory access Stage) forwarding unit

This forwarding unit keeps the eye on hazards happening in stage 4. The only instruction sequence that causes a hazard in this stage is the following.

```
lw x3, 8(x0)
sw x3, 8(x0)
```



The inputs and the outputs from the stage 4 forwarding unit is given below in the diagram.



stage 4 forwarding unit	
0	Regular value
1	stage 4 MEM READ

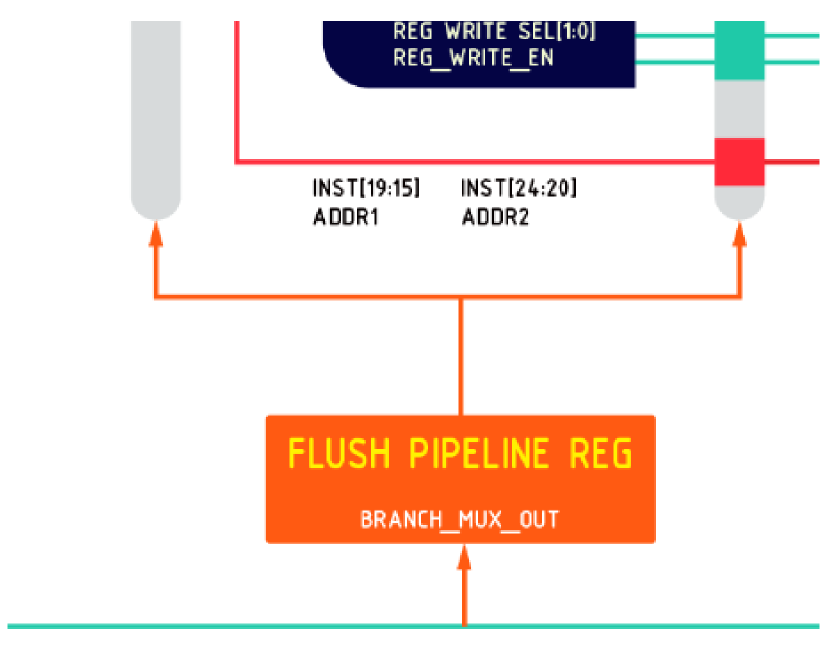
The only data dependency Hazard which cannot be handled by forwarding

The load use Hazard cannot be resolved by forwarding because when the data loading gets completed the following instruction is completed in his execution stage and moved to the memory access stage. We cannot handle this situation other than putting a nop instruction in between the instructions.

```
lw x3, 8(x0)
add x4, x3, x2
```

- ***Control Hazard Handling***

Control hazards occur when the instruction flow is diverted because of branching. In order to handle this hazard we have to flush a few pipeline stages. We have connected the first two pipeline registers flushing lines. If the branch control unit decides the instruction as a branch, then the first two pipeline registers get flushed so the two instructions loaded before the jump instructions are flushed.


REG WRITE SEL[1:0]
REG_WRITE_EN
INST[19:15]
ADDR1
INST[24:20]
ADDR2
FLUSH PIPELINE REG
BRANCH_MUX_OUT