Stall Unit

Synopsis:

The primary role of the Stall Unit within this code lies in the identification and management of stall conditions within a processor pipeline. Stalls, which manifest in the presence of data hazards, specifically read-after-write (RAW) hazards, prompt the Stall Unit to ensure that pipeline progression remains suspended until the detected hazard is effectively resolved.

Implementation:

Implemented as the **Fwd_Flush_Stall_Unit** module, the Stall Unit is endowed with diverse inputs related to branch prediction (**br_taken**), the current instruction (**inst**), the preceding instruction (**inst1**), and read-enable signals (**rd_en_1**). Its functional outputs encompass **stall** and **sel_rd1/sel_rd2**, which play pivotal roles in determining whether the pipeline should be stalled and which registers are to be read from.

Logic:

The logical framework of the module involves the computation of **raddr1**, **raddr2**, and **waddr** based on instruction fields. The module meticulously scrutinizes data hazards by juxtaposing **raddr1** and **raddr2** with **waddr**.

- Upon detecting a hazard, it promptly sets **stall** to 1, simultaneously ensuring that both **sel_rd1** and **sel_rd2** are set to 0.
- Conversely, in the absence of a hazard, **stall** is set to 0, and the determination of **sel_rd1** and **sel_rd2** hinges upon the matching of **raddr1** and **raddr2** with **waddr**.

```
    input  logic clk, stall;
    input  logic [31:0] new_inst;
    output logic [31:0] inst;

    always_ff@(posedge clk)
    begin
        if (|stall)
        begin
            inst <= inst;
        end
        else inst <= new_inst;
    end
endmodule
```

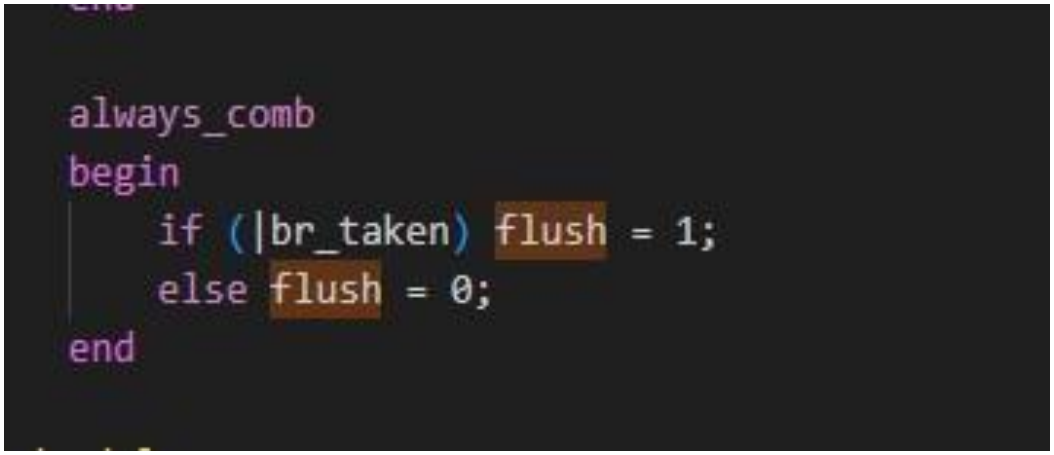
Flush Unit

Synopsis:

The Flush Unit assumes responsibility for managing branch mispredictions within the code, employing the strategy of flushing the pipeline when such mispredictions are detected. Upon detecting a branch misprediction, the Flush Unit discards instructions in the pipeline that should not have been executed.

Implementation:

The Flush Unit is also implemented within the **Fwd_Flush_Stall_Unit** module and outputs a single signal, **flush**, which is set to 1 when a branch misprediction is detected.



Logic:

The logical pathway of the Flush Unit entails an examination of the **br_taken** input to ascertain the occurrence of a branch misprediction.

- A true value of **br_taken** serves as an indicator of a misprediction, prompting the unit to set **flush** to 1.
- Conversely, if **br_taken** is false, signifying the absence of a misprediction, **flush** is diligently set to 0.

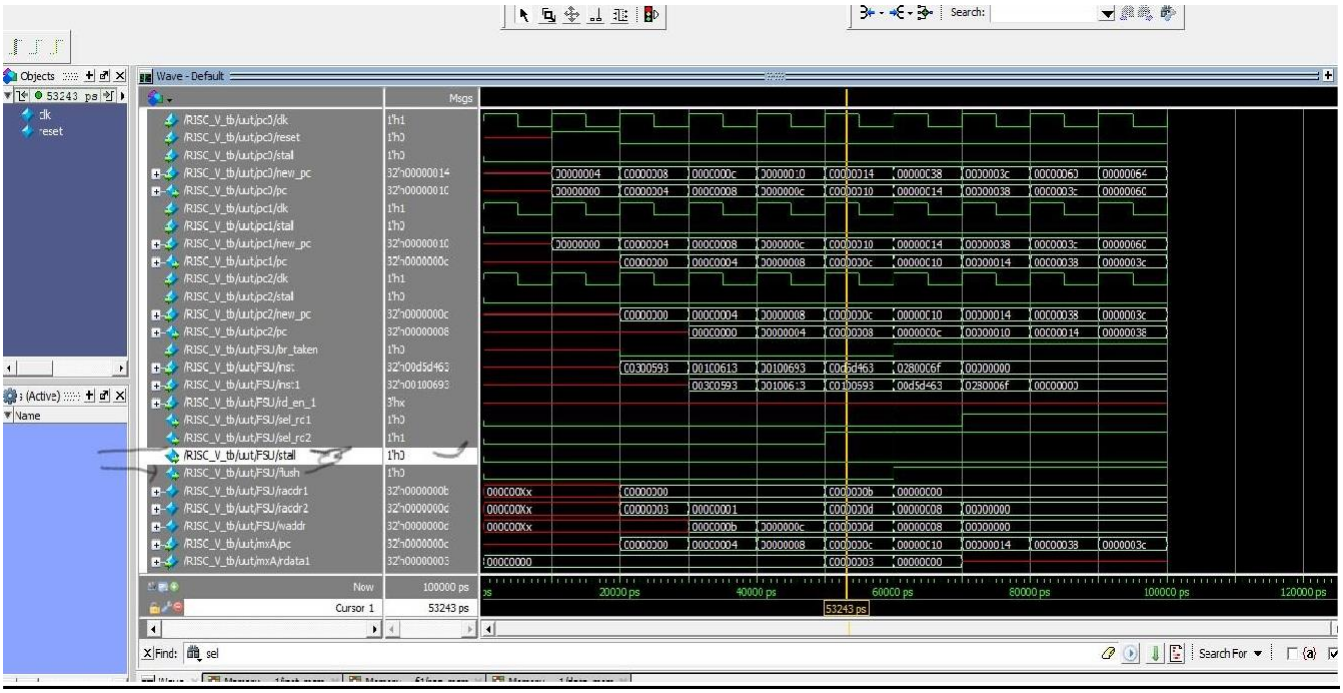
These three interdependent units, namely

- Stall,
- Flush,
- and Forwarding

– collaboratively ensure the seamless operation of the pipeline. They adeptly address or handle data hazards, facilitate data forwarding when feasible to mitigate stalls, and respond to branch mispredictions by executing pipeline flushes as necessitated. The associated diagrams in the test code within QuestaSim for factorial eloquently illustrate the absence of stalls and flushes during the code execution process.

Diagram:

Below is the shown diagram of stall and flush in test code. In the provided QuestaSim test code diagram for the factorial, there are no indications of stalls or flushes occurring during execution.



Forwarding Unit

Synopsis:

The Forwarding Unit in this code facilitates the forwarding of data from the execution stage to the decode stage in a processor pipeline. This functionality aids in resolving data hazards and enhancing pipeline performance by mitigating stalls.

Implementation:

Implemented within the **Fwd_Flush_Stall_Unit** module, the Forwarding Unit utilizes the same inputs and outputs as the Stall Unit but serves a distinct purpose: forwarding data to prevent stalls.

Logic:

Similar to the Stall Unit, the Forwarding Unit calculates **raddr1**, **raddr2**, and **waddr** based on instruction fields. It checks for data hazards, and in the absence of a hazard, sets **sel_rd1** and **sel_rd2** to 1 for registers matching **waddr**, facilitating data forwarding.

```
    begin
        stall = 0;
        if(raddr1 == waddr) sel_rd1 = 1;
        if(raddr2 == waddr) sel_rd2 = 1;
    end
nd
```

Diagram:

The accompanying diagram illustrates the forwarding process in the code.