MINI Project

30.04.2022

Punya Modi	2018AATS0336G
Mansi Doshi	2019A8PS0493G
Madhen Vyass Guru	2019AAPS0302G

Overview

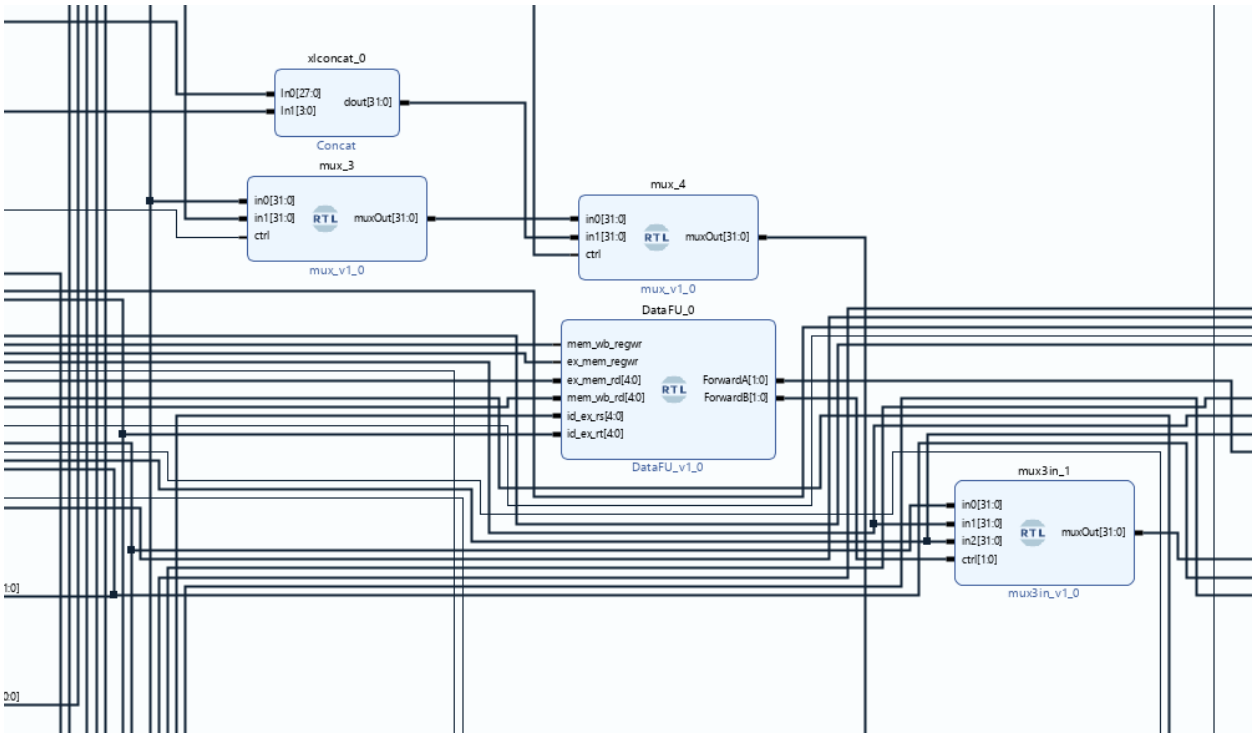
The aim of the project is to design a 5 staged pipeline MIPS processor that can both detect and correct data hazard and control hazard detection circuits by means of data forwarding and pipeline stalling

Approach

We have added circuits to enable data forwarding directly to the input of the ALU from both the EX/MEM and MEM/WB pipeline stage.

I. Data Hazards

Forwarding Unit:



This unit was added to forward data directly to the input of the ALU in two cases:

1. The 1st ALU operand is forwarded from the prior ALU result (1 instruction prior)
2. The 1st ALU operand is forwarded from the data memory of an earlier ALU result.(2 instructions prior)

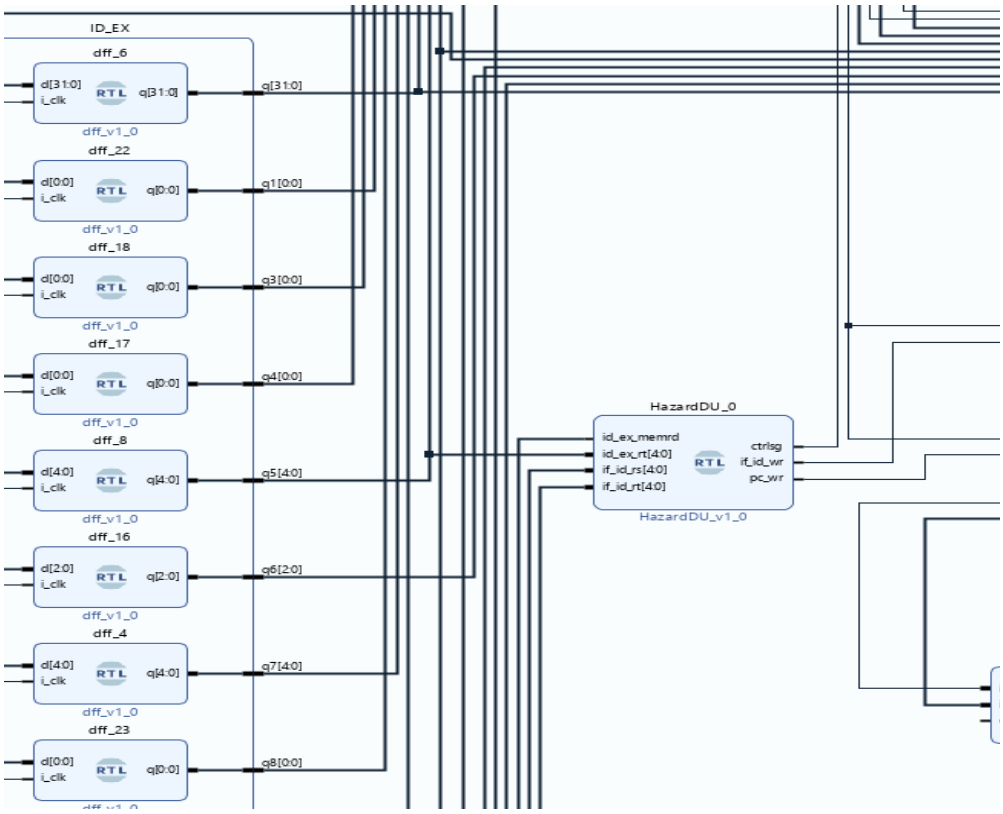
And this case should be specified both in the case of the 1st operand or the 2nd operand being involved in a hazard.

The code of the Data forwarding unit is specified below

```
module DataFU(
input mem_wb_regwr,
input ex_mem_regwr,
input [4:0] ex_mem_rd,
input [4:0] mem_wb_rd,
input [4:0] id_ex_rs,
input [4:0] id_ex_rt,
output reg [1:0] ForwardA,
output reg [1:0] ForwardB
);

always @(*)
begin
 ForwardA = 2'b00;
 if(ex_mem_regwr && ex_mem_rd != 'b00000 && ex_mem_rd == id_ex_rs)
  ForwardA = 2'b10;
 if(mem_wb_regwr && mem_wb_rd != 'b00000 && ex_mem_rd != id_ex_rs && mem_wb_rd == id_ex_rs)
  ForwardA = 2'b01;
end

always@(*)
begin
 ForwardB = 2'b00;
 if(ex_mem_regwr && ex_mem_rd != 'b00000 && ex_mem_rd == id_ex_rt)
  ForwardB = 2'b10;
 if(mem_wb_regwr && mem_wb_rd != 'b00000 && ex_mem_rd != id_ex_rt && mem_wb_rd == id_ex_rt)
  ForwardB = 2'b01;
 end

endmodule
```

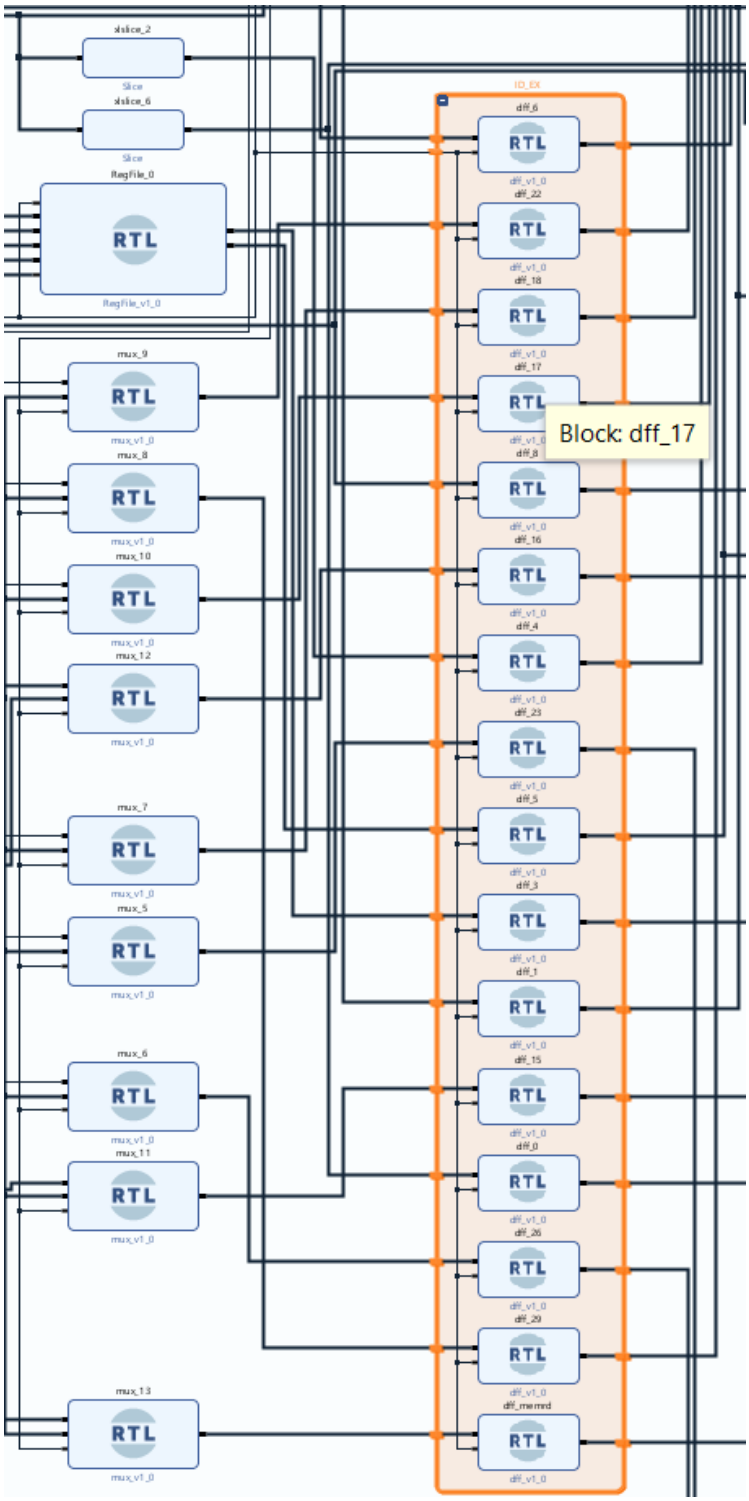
To abide by MIPS condition that every use of $0 as an operand must yield an operand value of zero, we don't forward results that are destined for $0. The other conditions are matching the operand and destination registers at different stages of the pipeline if/when they overlap.

Hazard Detection Unit:



This unit was specifically added to cater to the case when an instruction tries to read a register following a load instruction. In this case, the data is still being obtained from the memory in the penultimate clock cycle while the ALU is simultaneously performing the next instruction.

In this case, we are required to stall the pipeline for a minimum of 1 clock cycle, which we have done by gating the clock of PC(Program Counter) and the IF/ID pipeline stage and by forcing 0 to all 9 control signals, to ensure that no data is being written into the register file/ data memory. This stall is effectively a nop (a temporary bubble) being placed in the pipeline so that the data dependencies of the following instructions can be satisfied.



These series of muxes works in tandem with the Hazard Detection Unit to ensure that all the control signals have a value of 0 if the load hazard condition is true.

We implemented the Hazard detection unit as follows -

```
module HazardDU(
input id_ex_memrd,
input [4:0] id_ex_rt,
input [4:0] if_id_rs,
input [4:0] if_id_rt,
output reg ctrlsg,
output reg if_id_wr,
output reg pc_wr
);

always @(*)
begin
 ctrlsg = 1'b0;
 if_id_wr = 1'b1;
 pc_wr = 1'b1;

 if(id_ex_memrd && ((id_ex_rt == if_id_rs)||(id_ex_rt == if_id_rt)))
 ctrlsg = 1'b1;
 if_id_wr = 1'b0;
 pc_wr = 1'b0;
end
endmodule
```

Similarly we Implemented the Data Forwarding Unit as follows -

```
module DataFU(
input mem_wb_regwr,
input ex_mem_regwr,
input [4:0] ex_mem_rd,
input [4:0] mem_wb_rd,
input [4:0] id_ex_rs,
input [4:0] id_ex_rt,
output reg [1:0] ForwardA,
output reg [1:0] ForwardB
);

always @(*)
begin
 ForwardA = 2'b00;
 if(ex_mem_regwr && ex_mem_rd != 'b00000 && ex_mem_rd == id_ex_rs)
  ForwardA = 2'b10;
 if(mem_wb_regwr && mem_wb_rd != 'b00000 && ex_mem_rd != id_ex_rs && mem_wb_rd == id_ex_rs)
  ForwardA = 2'b01;
end

always@(*)
begin
 ForwardB = 2'b00;
 if(ex_mem_regwr && ex_mem_rd != 'b00000 && ex_mem_rd == id_ex_rt)
  ForwardB = 2'b10;
 if(mem_wb_regwr && mem_wb_rd != 'b00000 && ex_mem_rd != id_ex_rt && mem_wb_rd == id_ex_rt)
  ForwardB = 2'b01;
 end

endmodule
```

II. Control Hazards

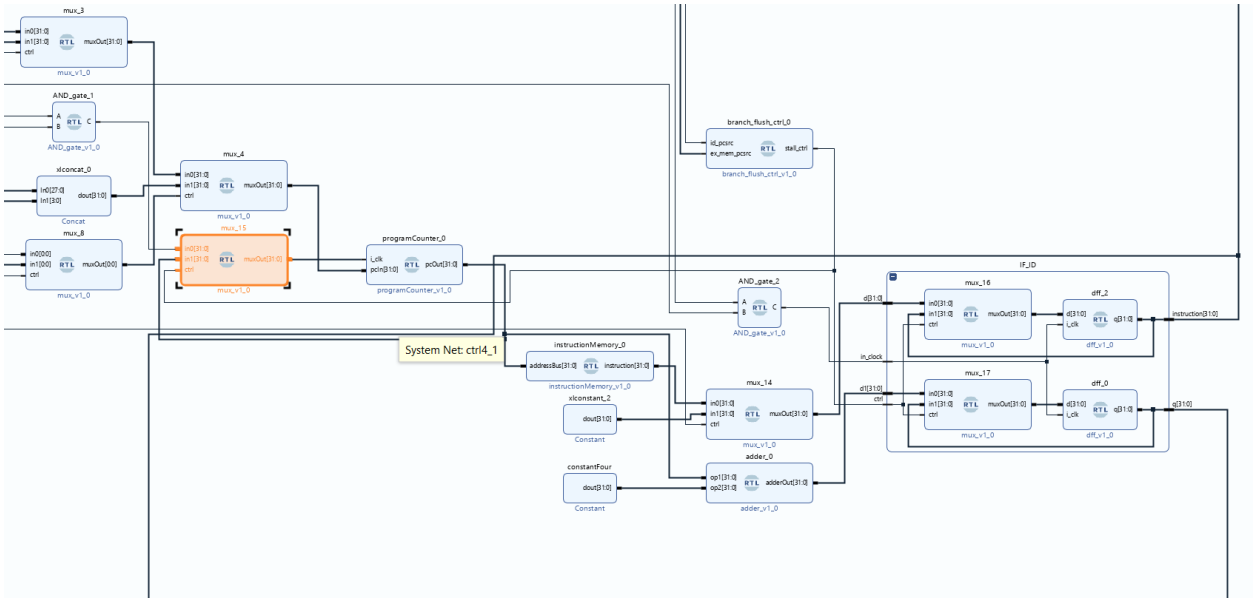
Control Hazards cover the pipeline instructions which involve branches. There are two such instructions we are working with here.

1. Jump
2. Branch

For a jump instruction, we immediately flush the IF/ID pipeline with 0, and we have moved the jump address calculation to the ID stage (removed the intermediary flip flops) to allow for the jump within 1 clock cycle.

For a branch instruction, we have chosen to stall the pipeline (PC and IF/ID) for 2 clock cycles to allow for the branch address to be calculated/ branch to be confirmed with the zero flag of ALU. We have created a new module to create the control signal for this stall(branch_flush_ctrl). This control signal is dictated by the pcsrc signal at the ID stage of the pipeline and the MEM stage of the pipeline. (When both of them are equal, it means the address for the branch has been calculated)

The following images are the components added for the processor to deal with control hazards



Here we have given the following instructions to the processor -

```
module instructionMemory(
input   [31:0] addressBus,
output  [31:0] instruction
);

reg [31:0] mem [1023:0];

initial
begin
    mem[0] = 32'h20090064;//addi $t1,$zero,100
    mem[1] = 32'h200b0005;//addi $t3,$zero,5
    mem[2] = 32'h200d0007;//add $t5,$zero,7
    mem[3] = 32'h200e000c;//add $t6,$zero,12
    mem[4] = 32'h012b5022;//sub $ts,$t1,$t3
    mem[5] = 32'h014dc824;//and $t9,$t2,$t5
    mem[6] = 32'h01ca7825;//or $t7,$t6,$t2
    mem[7] = 32'h014a6020;//add $t4,$t2,$t2
end

assign instruction =  mem[addressBus[31:2]];


endmodule
```

The following screenshots are the output result of the simulation based on the input instructions given above -