ECE 586 - Computer Architecture

Project report
On
Design of 5-stage MIPS Pipelined Processor

By Team 31
Adithya Rajagopal
Swaroop Nellur
Preetam Basti

CONTENTS

I. Objective:

To develop an execution-driven MIPS pipeline simulator. The simulator will take the provided memory_image.txt as an input.
It will implement two key features:

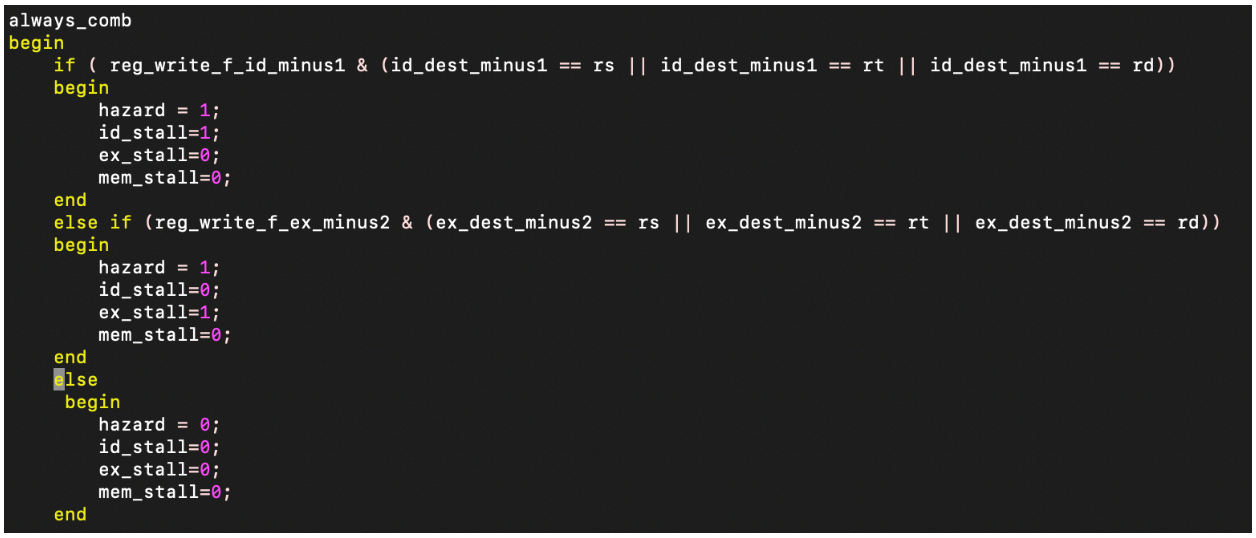
i) Functional Simulator which simulates the MIPS-lite ISA and captures the impact of instruction execution in machine state. This need not be cycle accurate simulations.

ii) A timing simulator which shows timing details of the 5-stage pipeline.

II. Implementation:

We implemented MIPS-lite design using SystemVerilog language on Questa Sim.

Below Code is the conditions to generate the hazard where the present address of all the registers are compared with destination registers for the previous 2 instructions. Only when register write is present.

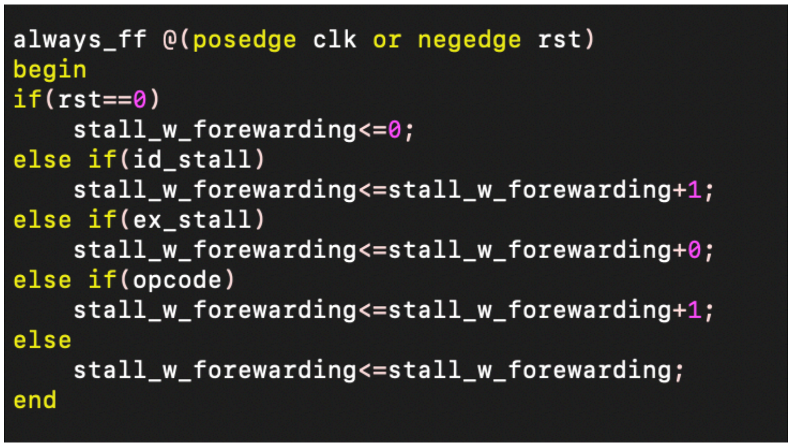


```
always_comb
begin
    if ( reg_write_f_id_minus1 & (id_dest_minus1 == rs || id_dest_minus1 == rt || id_dest_minus1 == rd))
    begin
        hazard = 1;
        id_stall=1;
        ex_stall=0;
        mem_stall=0;
    end
    else if (reg_write_f_ex_minus2 & (ex_dest_minus2 == rs || ex_dest_minus2 == rt || ex_dest_minus2 == rd))
    begin
        hazard = 1;
        id_stall=0;
        ex_stall=1;
        mem_stall=0;
    end
    else
     begin
        hazard = 0;
        id_stall=0;
        ex_stall=0;
        mem_stall=0;
    end
```

After this we have different conditions on stall count to be incremented.

Stall conditions for forwarding case:

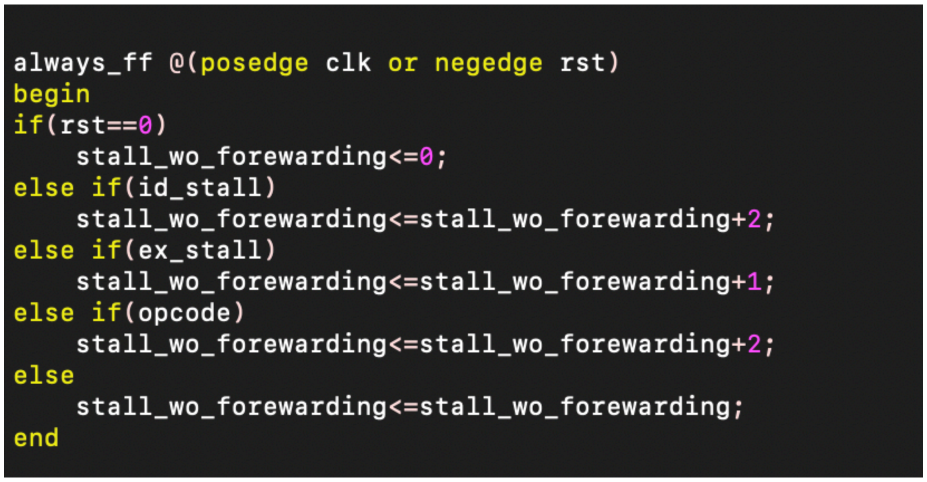
- There will be no delays if the dependent instruction comes to right after the procedure instruction. As a result, the stall penalty is 0.
- The stall penalty is one if the dependent instruction comes right after the producer instruction and the producer instruction is a LOAD instruction
- The stall penalty is 2 if the instruction is a branch and is taken, or if the instruction jump register is executed
- In the below code The stall count is incremented only when there is hazard in the previous instruction.



```
always_ff @(posedge clk or negedge rst)
begin
if(rst==0)
    stall_w_forewarding<=0;
else if(id_stall)
    stall_w_forewarding<=stall_w_forewarding+1;
else if(ex_stall)
    stall_w_forewarding<=stall_w_forewarding+0;
else if(opcode)
    stall_w_forewarding<=stall_w_forewarding+1;
else
    stall_w_forewarding<=stall_w_forewarding;
end
```

Stall condition for no-forwarding case:

- The stall penalty is 2 cycles if the dependent instruction comes shortly after the producer instruction
- The stall penalty is 1 cycle if an intermediate instruction separates the producer and dependent instructions
- If there are two instructions between the procedure and the dependent instruction and there is no dependence between them, the stall penalty is 0.
- The stall penalty is 2 if the instruction is a branch and its taken, or if the jump register is executed.
- Below code will increment stall count by 2 when there is a hazard in the ID stage.



```
always_ff @(posedge clk or negedge rst)
begin
if(rst==0)
    stall_wo_forewarding<=0;
else if(id_stall)
    stall_wo_forewarding<=stall_wo_forewarding+2;
else if(ex_stall)
    stall_wo_forewarding<=stall_wo_forewarding+1;
else if(opcode)
    stall_wo_forewarding<=stall_wo_forewarding+2;
else
    stall_wo_forewarding<=stall_wo_forewarding;
end
```

III. Simulation Results:

Test results of transcript for all 3 conditions are pasted below which highlights all the required control registers and data registers.

Registers	**Values(decimals)**
PC	112
Total number of instructions	911
Arithmetic Instructions	375
Logical instructions	61
Memory access instructions	300
Control transfer instructions	175
R11	1044
R12	1836
R13	2640
R14	25
R15	-188
R16	213
R17	29
R18	3440

R19	-1
R20	-2
R21	-1
R22	76
R23	3
R24	-1
R25	3
Total number of clock cycles without forwarding	1586
Number of stalls without forwarding	669
Total number of clock cycles with forwarding	1218
Number of stalls with forwarding	301
Speedup	1.302135

Below is the transcript of the memory_image.txt

```
# //  Questa Sim-64
# //  Version 10.6b linux_x86_64 May 25 2017
# //
# //  Copyright 1991-2017 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  QuestaSim and its associated documentation contain trade
# //  secrets and commercial or financial information that are the
property of
# //  Mentor Graphics Corporation and are privileged, confidential,
# //  and exempt from disclosure under the Freedom of Information Act,
# //  5 U.S.C. Section 552. Furthermore, this information
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //
# vsim work.tb -voptargs="+acc" "+MEM_IMAGE=memory_image.txt" -do "add
wave sim:/tb/i_main/i_inst_fetch/* ; add wave sim:/tb/i_main/i_decode/*
;add wave sim:/tb/i_main/i_ex/* ;add wave sim:/tb/i_main/i_memory/* ;add
wave sim:/tb/i_main/i_writeback/* ;add wave sim:/tb/*;run -all"
# Start time: 14:33:41 on Jun 08,2023
# ** Note: (vsim-3812) Design is being optimized...
# Loading sv_std.std
# Loading work.testbench_sv_unit(fast)
# Loading work.tb(fast)
# Loading work.main_sv_unit(fast)
# Loading work.main(fast)
# Loading work.if_sv_unit(fast)
# Loading work.inst_f(fast)
# Loading work.id_sv_unit(fast)
# Loading work.id(fast)
# Loading work.alu_sv_unit(fast)
# Loading work.alu(fast)
# Loading work.mem_sv_unit(fast)
# Loading work.mem(fast)
# Loading work.wb_sv_unit(fast)
# Loading work.wb(fast)
# ** Warning: (vsim-PLI-3691) ../rtl/if.sv(98): Expected a system task,
not a system function '$value$plusargs'.
#    Time: 0 ns  Iteration: 0  Instance: /tb/i_main/i_inst_fetch File:
../rtl/if.sv
# ** Warning: (vsim-PLI-3691) ../rtl/mem.sv(29): Expected a system task,
not a system function '$value$plusargs'.
#    Time: 0 ns  Iteration: 0  Instance: /tb/i_main/i_memory File:
../rtl/mem.sv
# add wave sim:/tb/i_main/i_inst_fetch/*
#  add wave sim:/tb/i_main/i_decode/*
# add wave sim:/tb/i_main/i_ex/*
# add wave sim:/tb/i_main/i_memory/*
# add wave sim:/tb/i_main/i_writeback/*
# add wave sim:/tb/*
# run -all
# ..............Instruction Counts...............
# Total number of instructions:            911
# Arithmetic instructions:              375
# Logical instructions:                  61
# Memory access instructions:           300
# Control transfer instructions:        175
# ..............Final Register State.............
# Program counter:        112
# R1:           0
# R2:           0
# R3:           0
# R4:           0
# R5:           0
# R6:           0
# R7:           0
# R8:           0
# R9:           0
# R10:          0
# R11:       1044
# R12:       1836
# R13:       2640
# R14:         25
# R15:       -188
# R16:        213
# R17:         29
# R18:       3440
# R19:         -1
# R20:         -2
# R21:         -1
# R22:         76
# R23:          3
# R24:         -1
# R25:          3
# R26:          0
# R27:          0
# R28:          0
# R29:          0
# R30:          0
# R31:          0
# ..........Final memory state...........
# Address:        2400, Contents:           2
# Address:        2404, Contents:           4
# Address:        2408, Contents:           6
# Address:        2412, Contents:           8
# Address:        2416, Contents:          10
# Address:        2420, Contents:          12
# Address:        2424, Contents:          14
# Address:        2428, Contents:          16
# Address:        2432, Contents:          18
# Address:        2436, Contents:          29
# Address:        2440, Contents:          22
# Address:        2444, Contents:          24
# Address:        2448, Contents:          26
# Address:        2452, Contents:          28
# Address:        2456, Contents:          30
# Address:        2460, Contents:          32
# Address:        2464, Contents:          34
# Address:        2468, Contents:          36
# Address:        2472, Contents:          38
# Address:        2476, Contents:          59
# Address:        2480, Contents:          42
# Address:        2484, Contents:          44
# Address:        2488, Contents:          46
# Address:        2492, Contents:          48
# Address:        2496, Contents:          50
# Address:        2500, Contents:          52
# Address:        2504, Contents:          54
# Address:        2508, Contents:          56
# Address:        2512, Contents:          58
# Address:        2516, Contents:          89
# Address:        2520, Contents:          62
# Address:        2524, Contents:          64
# Address:        2528, Contents:          66
# Address:        2532, Contents:          68
# Address:        2536, Contents:          70
# Address:        2540, Contents:          72
# Address:        2544, Contents:          74
# Address:        2548, Contents:          76
# Address:        2552, Contents:          78
# Address:        2556, Contents:         119
# Address:        2560, Contents:          82
# Address:        2564, Contents:          84
# Address:        2568, Contents:          86
# Address:        2572, Contents:          88
# Address:        2576, Contents:          90
# Address:        2580, Contents:          92
# Address:        2584, Contents:          94
# Address:        2588, Contents:          96
# Address:        2592, Contents:          98
# Address:        2596, Contents:         149
# Address:        2600, Contents:           2
# Address:        2604, Contents:           4
# Address:        2608, Contents:           6
# Address:        2612, Contents:           8
# Address:        2616, Contents:          10
# Address:        2620, Contents:          12
# Address:        2624, Contents:          14
# Address:        2628, Contents:          16
# Address:        2632, Contents:          18
# Address:        2636, Contents:          29
# ..........Timing Simulator without forwarding..........
# Total number of clock cycles:       1586
# ..........Timing Simulator with forwarding..........
# Total number of clock cycles:       1218
# ..........Speedup overall..........
# Speedup overall : 1.302135
# Program Halted
# ** Note: $finish    : ../rtl/testbench.sv(206)
#    Time: 7350 ns  Iteration: 0  Instance: /tb
# 1
# Break in Module tb at ../rtl/testbench.sv line 206
# End time: 14:43:37 on Jun 08,2023, Elapsed time: 0:09:56
# Errors: 0, Warnings: 2
```

IV. Responsibilities

	Adithya	**Swaroop**	**Preetam**
Task 1	Coded Instruction Decode	Coded ALU	Coded Instruction Fetch
Task 2	Coded Memory	Coded Execute	Coded Writeback
Task 3	Resolved pipeline errors(X prop)	Simulation results	Memory image read
Task 4	Integration errors	Writing Multiple Memory_image.txt for testing	Hazard implementation
Task 5	Coded hazard requirements from all the blocks to fetch previous inst	Stall Count actual logic	Debug on multiple memory_image.txt
Task 6	Documentation		

V. Future Tasks

Do not hardcode with the name memory_image file.