MP4.1 Report

Progress Report

For checkpoint 1, we distributed the checkpoint objectives as follows…

- **Neo -** I worked on the control ROM and developed the unit tests to test the design.
- **Naveen / Anchit -** We worked together to design the pipeline stages and instantiated the stages and the relevant modules within the datapath module.

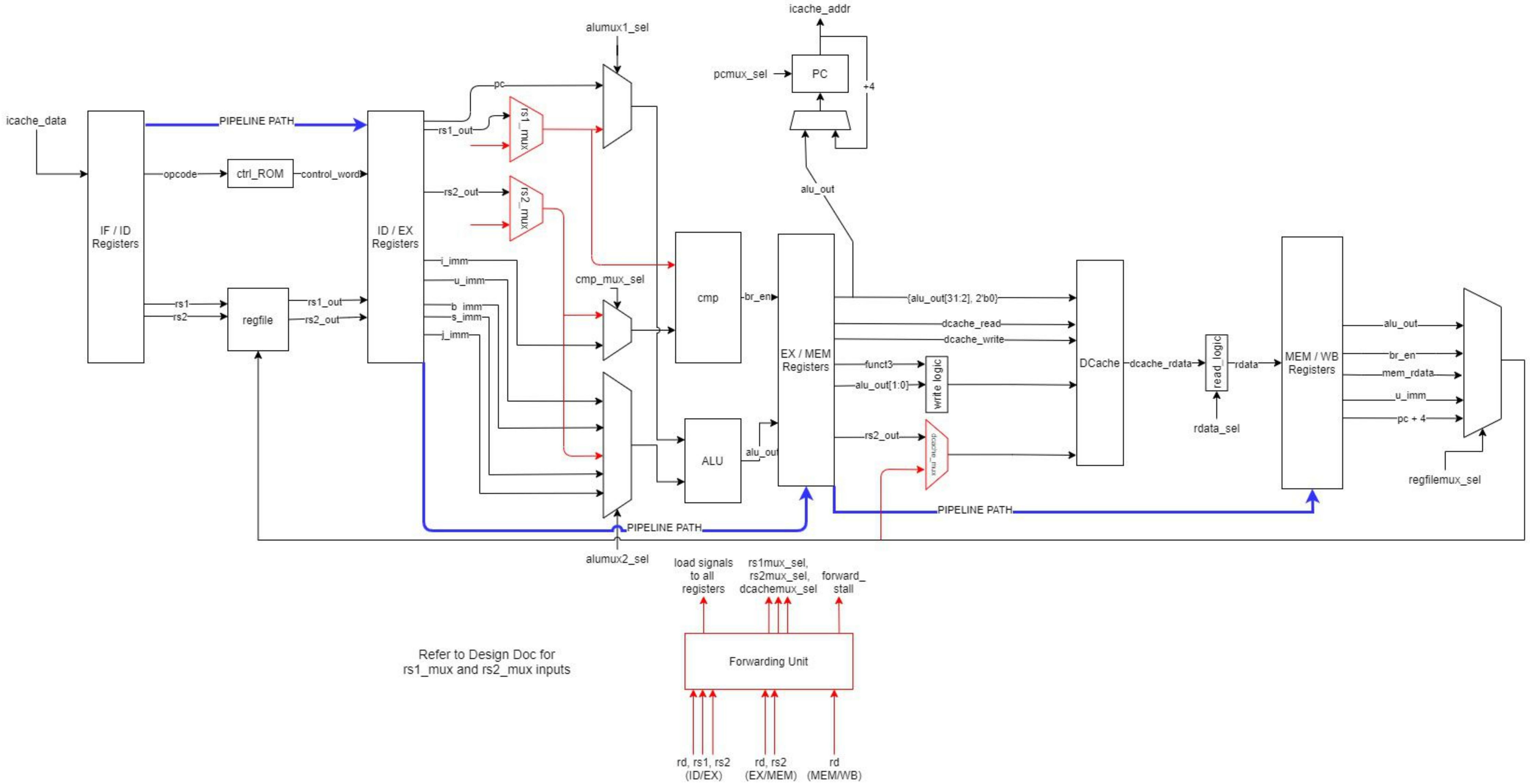
For this checkpoint, we first tested our design using the testbench given, testing the pipeline and that the proper result was stored after it was written back to the register. Furthermore, we implemented unit tests covering different types of instructions such as load/stores, conditional and unconditional jumps, and register and immediate operations.

Roadmap

For the next checkpoint, we have decided to split up the following tasks among us three.

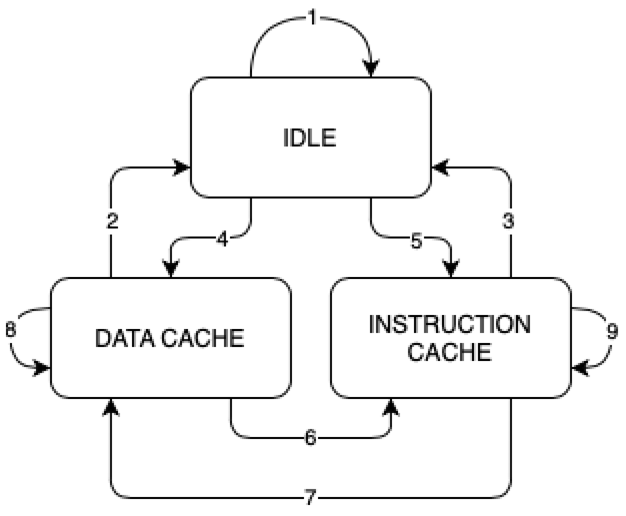
- **Neo -** Forwarding
- **Anchit** - Data Hazards / Stalling
- **Naveen -** Arbiter

However, we all worked together to design the implementation and talk through the edge cases.



Refer to Design Doc for rs1_mux and rs2_mux inputs

Arbiter State Machine

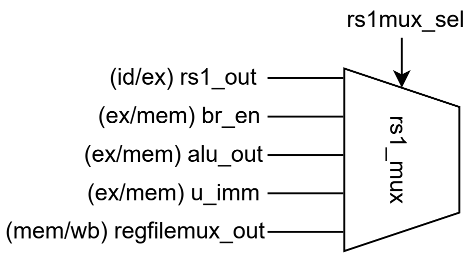


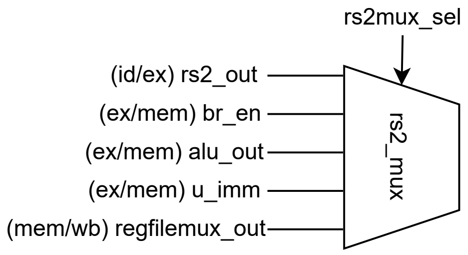
1	(dpmem_read == 1’b0 & dpmem_write == 1’b0) & (ipmem_read == 1’b0 & ipmem_write == 1’b0)
2	(dpmem_read == 1’b0 & dpmem_write == 1’b0) & (ipmem_read == 1’b0 & ipmem_write == 1’b0)
3	(dpmem_read == 1’b0 & dpmem_write == 1’b0) & (ipmem_read == 1’b0 & ipmem_write == 1’b0)
4	(dpmem_read == 1’b1 | dpmem_write == 1’b1) & (ipmem_read == 1’b0 & ipmem_write == 1’b0)
5	(ipmem_read == 1’b1 | ipmem_write == 1’b1)
6	(dpmem_read == 1’b0 & dpmem_write == 1’b0) & (ipmem_read == 1’b1 | ipmem_write = 1’b1) & (resp_o == 1’b1)

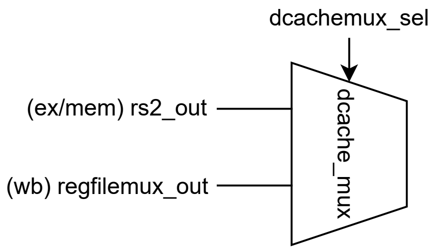
Arbiter State Machine

7	(ipmem_read == 1'b0 & ipmem_write == 1'b0) & (dpmem_read == 1'b1 | dpmem_write == 1'b1) & (resp_o == 1'b1)
8	(dpmem_read == 1'b1 | dpmem_write == 1'b1) & (ipmem_read == 1'b0 & ipmem_write == 1'b0) & (resp_o == 1'b1)
9	(ipmem_read == 1'b1 | ipmem_write == 1'b1) & (resp_o == 1'b1)

Forwarding Muxes and Logic







RS1MUX_SEL LOGIC

```
if (idex_rs1 == exmem_rd and
exmem_load_regfile == 1):
    if (idex_opcode == lui):
        rs1mux_sel = exmem_u_imm
    else if (idex_opcode == slt/sltu):
        rs1mux_sel = exmem_br_en
    else:
        rs1mux_sel = exmem_alu_out
else if (idex_rs1 == memwb_rd and
memwb_load_regfile == 1):
    rs1mux_sel = wb_regfilemux_out
else:
    rs1mux_sel = rs1_out
```

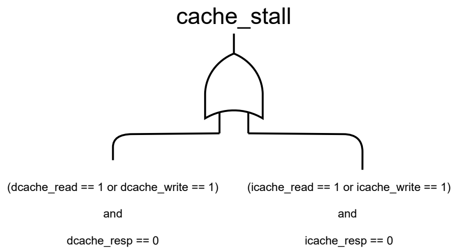
RS2MUX_SEL LOGIC

```
if (idex_rs2 == exmem_rd and
exmem_load_regfile == 1):
    if (idex_opcode == lui):
        rs2mux_sel = exmem_u_imm
    else if (idex_opcode == slt/sltu):
        rs2mux_sel = exmem_br_en
    else:
        rs2mux_sel = exmem_alu_out
else if (idex_rs1 == memwb_rd and
memwb_load_regfile == 1):
    rs2mux_sel = wb_regfilemux_out
else:
    rs2mux_sel = rs2_out
```

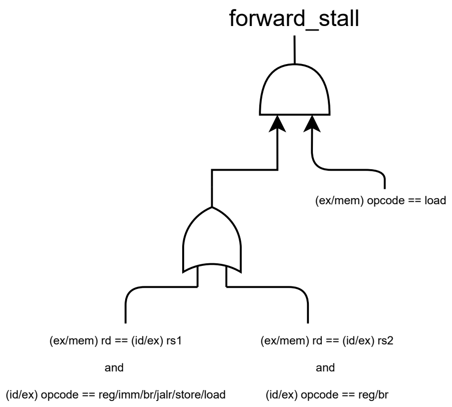
DCACHEMUX_SEL LOGIC

```
if (memwb_rd == exmem_rs2 and
memwb_load_regfile == 1):
    dcachemux_sel = wb_regfilemux_out
else:
    dcachemux_sel = exmem_rs2_out
```

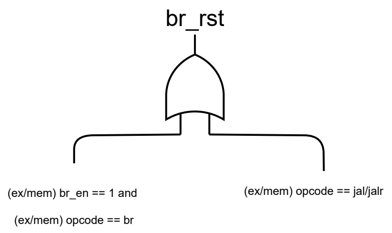
Cache Stall


cache_stall
(dcache_read == 1 or dcache_write == 1)
and
dcache_resp == 0
(icache_read == 1 or icache_write == 1)
and
icache_resp == 0

Forward Stall


forward_stall
(ex/mem) opcode == load
(ex/mem) rd == (id/ex) rs1
and
(id/ex) opcode == reg/imm/br/jalr/store/load
(ex/mem) rd == (id/ex) rs2
and
(id/ex) opcode == reg/br

Branch Reset



Load and Reset Logic

pc_load = (~cache_stall) | (~forward_stall)
ifid_load = (~cache_stall) | (~forward_stall)
idex_load = (~cache_stall) | (~forward_stall)
exmem_load = (~cache_stall)
memwb_load = (~cache_stall)

pc_rst = rst
ifid_rst = rst | br_rst
idex_rst = rst | br_rst
exmem_rst = rst | forward_stall
memwb_rst = rst