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.

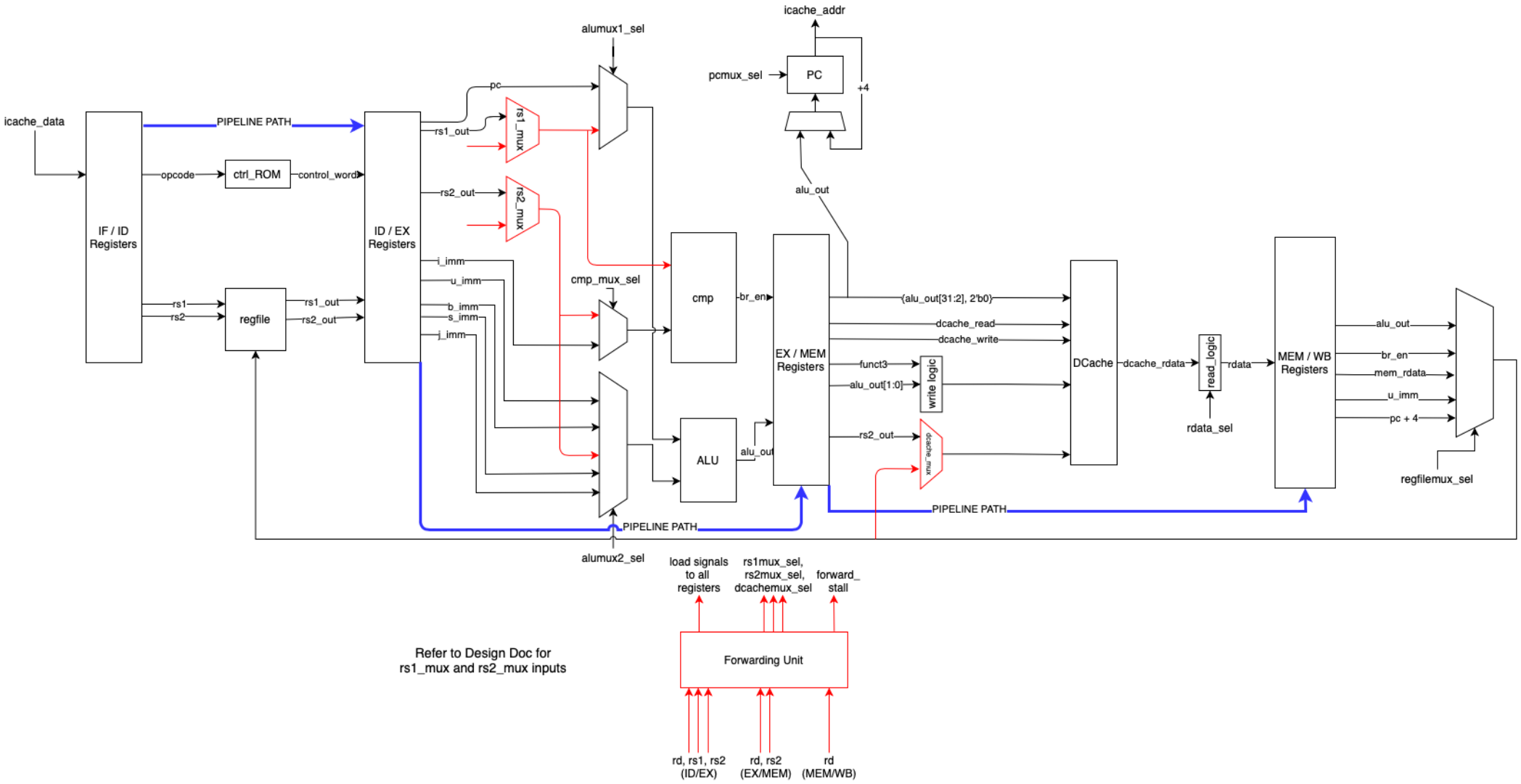
For this checkpoint, we first tested our design using the testbench given, testing the pipeline and that the proper result was stored after it was written back to the register. Furthermore, we implemented unit tests covering different types of instructions such as load/stores, conditional and unconditional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

Roadmap

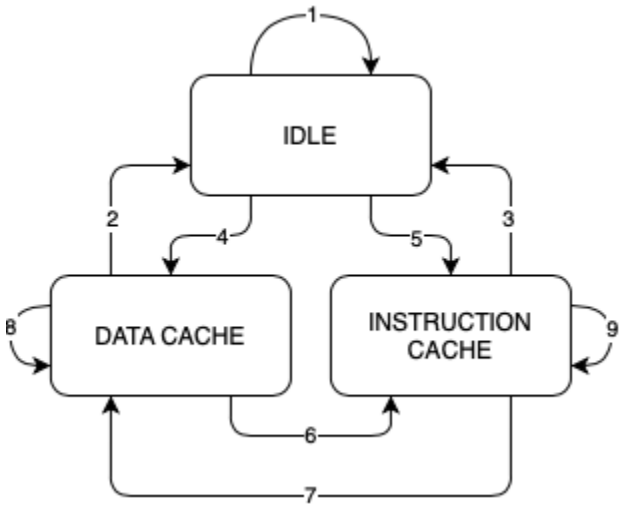
For the next checkpoint, we have decided to split up the following tasks among us three.

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

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


icache_addr
alumux1_sel
pcmux_sel
PC
+4
icache_data
PIPELINE PATH
pc
rs1_out
rs1_mux
rs2_out
rs2_mux
alu_out
opcode
ctrl_ROM
control_word
IF / ID Registers
ID / EX Registers
i_imm
u_imm
b_imm
s_imm
j_imm
cmp_mux_sel
cmp
br_en
rs1
rs2
regfile
rs1_out
rs2_out
EX / MEM Registers
{alu_out[31:2], 2'b0}
dcache_read
dcache_write
funct3
alu_out[1:0]
write logic
rs2_out
dcache_mux
DCache
dcache_rdata
read_logic
rdata
rdata_sel
MEM / WB Registers
alu_out
br_en
mem_rdata
u_imm
pc + 4
regfilemux_sel
ALU
alu_out
PIPELINE PATH
PIPELINE PATH
alumux2_sel
load signals to all registers
rs1mux_sel, rs2mux_sel, dcachemux_sel
forward_ stall
Forwarding Unit
rd, rs1, rs2 (ID/EX)
rd, rs2 (EX/MEM)
rd (MEM/WB)
Refer to Design Doc for rs1_mux and rs2_mux inputs

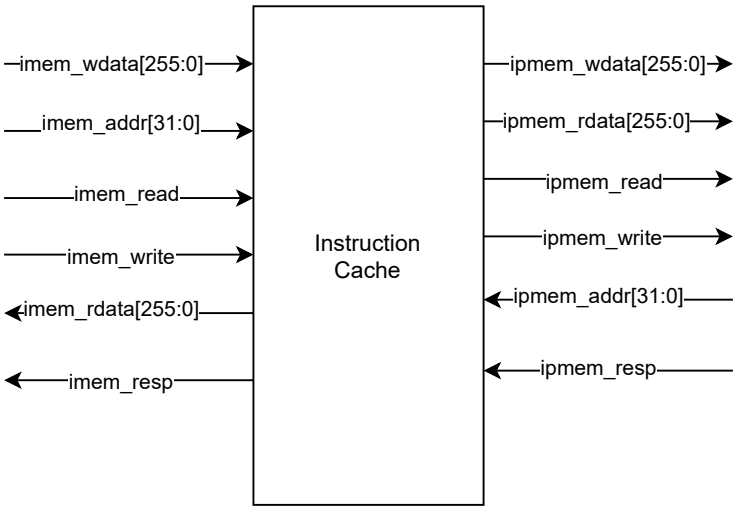
Arbiter State Machine

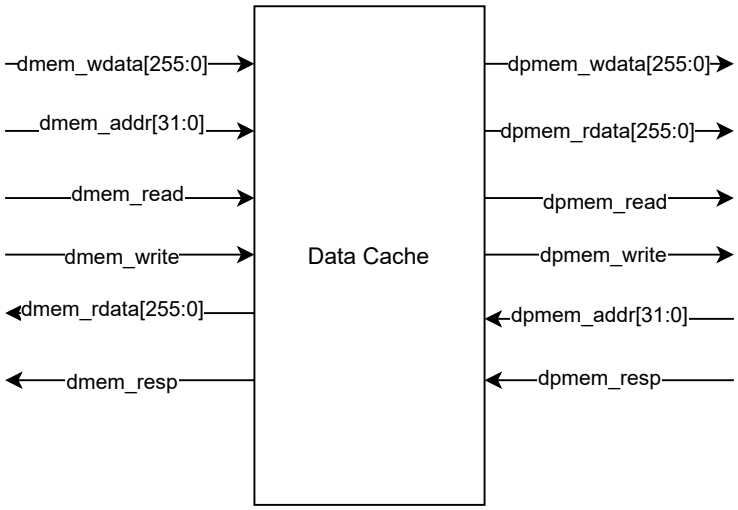


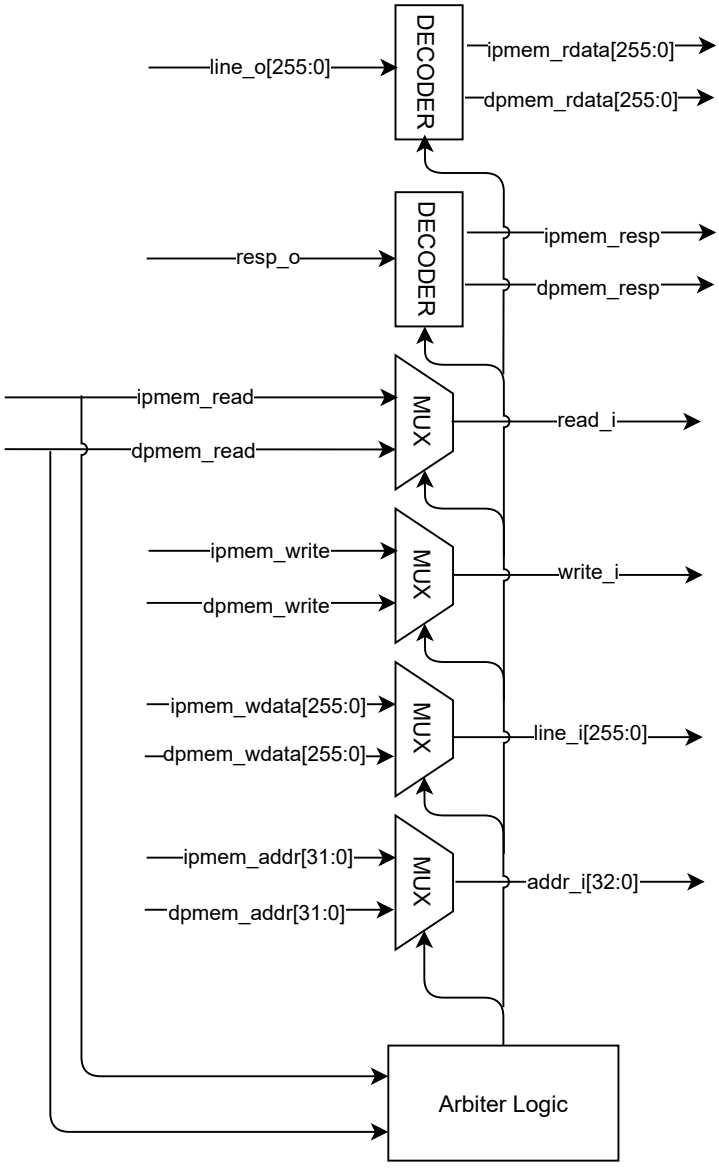
1	(dpmem_read == 1'b0 & dpmem_write == 1'b0) & (ipmem_read == 1'b0 & ipmem_write == 1'b0)
2	(dpmem_read == 1'b0 & dpmem_write == 1'b0) & (ipmem_read == 1'b0 & ipmem_write == 1'b0)
3	(dpmem_read == 1'b0 & dpmem_write == 1'b0) & (ipmem_read == 1'b0 & ipmem_write == 1'b0)
4	(dpmem_read == 1'b1 | dpmem_write == 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)

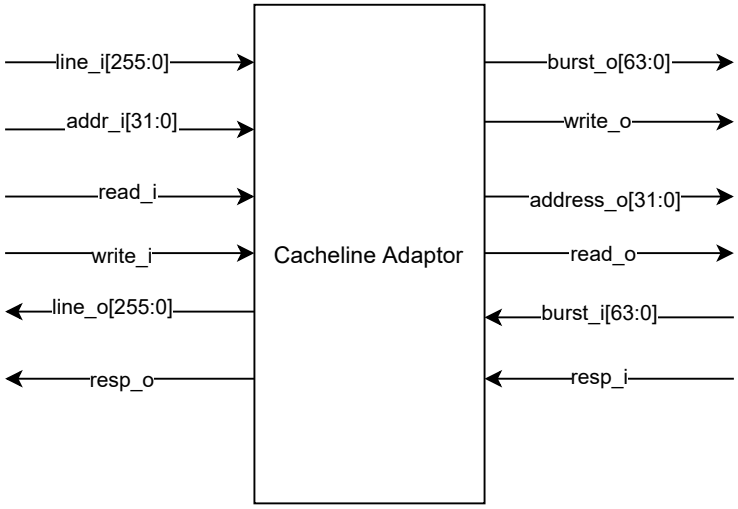
Arbiter State Machine

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)

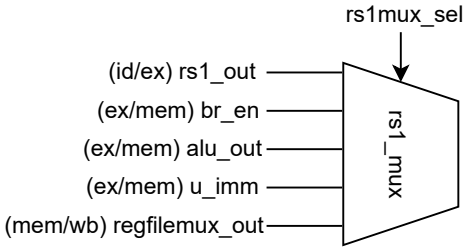

imem_wdata[255:0]
imem_addr[31:0]
imem_read
imem_write
imem_rdata[255:0]
imem_resp
Instruction Cache
ipmem_wdata[255:0]
ipmem_rdata[255:0]
ipmem_read
ipmem_write
ipmem_addr[31:0]
ipmem_resp

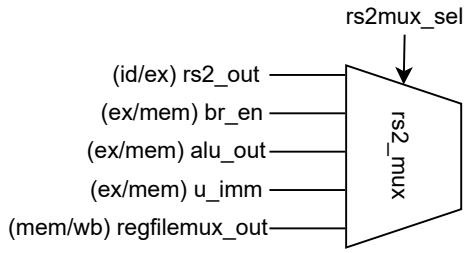

dmem_wdata[255:0]
dmem_addr[31:0]
dmem_read
dmem_write
dmem_rdata[255:0]
dmem_resp
Data Cache
dpmem_wdata[255:0]
dpmem_rdata[255:0]
dpmem_read
dpmem_write
dpmem_addr[31:0]
dpmem_resp

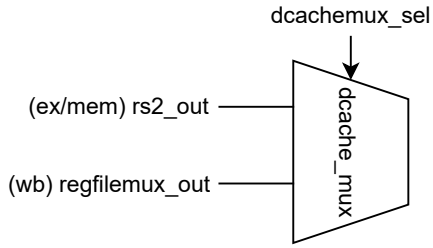

line_o[255:0]
DECODER
ipmem_rdata[255:0]
dpmem_rdata[255:0]
resp_o
DECODER
ipmem_resp
dpmem_resp
ipmem_read
dpmem_read
MUX
read_i
ipmem_write
dpmem_write
MUX
write_i
ipmem_wdata[255:0]
dpmem_wdata[255:0]
MUX
line_i[255:0]
ipmem_addr[31:0]
dpmem_addr[31:0]
MUX
addr_i[32:0]
Arbiter Logic


line_i[255:0]
addr_i[31:0]
read_i
write_i
line_o[255:0]
resp_o
Cacheline Adaptor
burst_o[63:0]
write_o
address_o[31:0]
read_o
burst_i[63:0]
resp_i

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
```

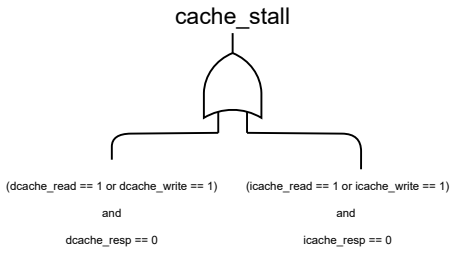
RS2MUX_SEL LOGIC

```
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
```

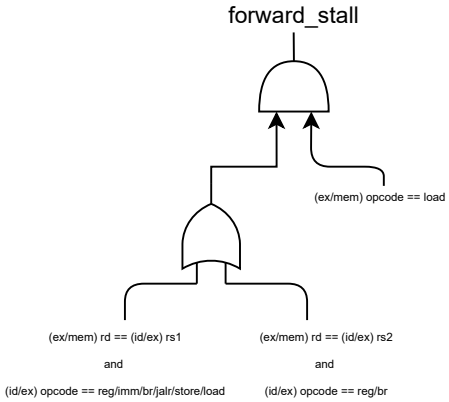
DCACHEMUX_SEL LOGIC

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

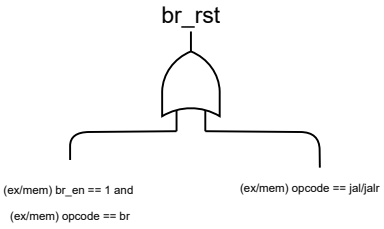
Cache Stall



Forward Stall



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_load = rst
ifid_load = rst | br_rst
idex_load = rst | br_rst
exmem_load = rst | forward_stall
memwb_load = rst