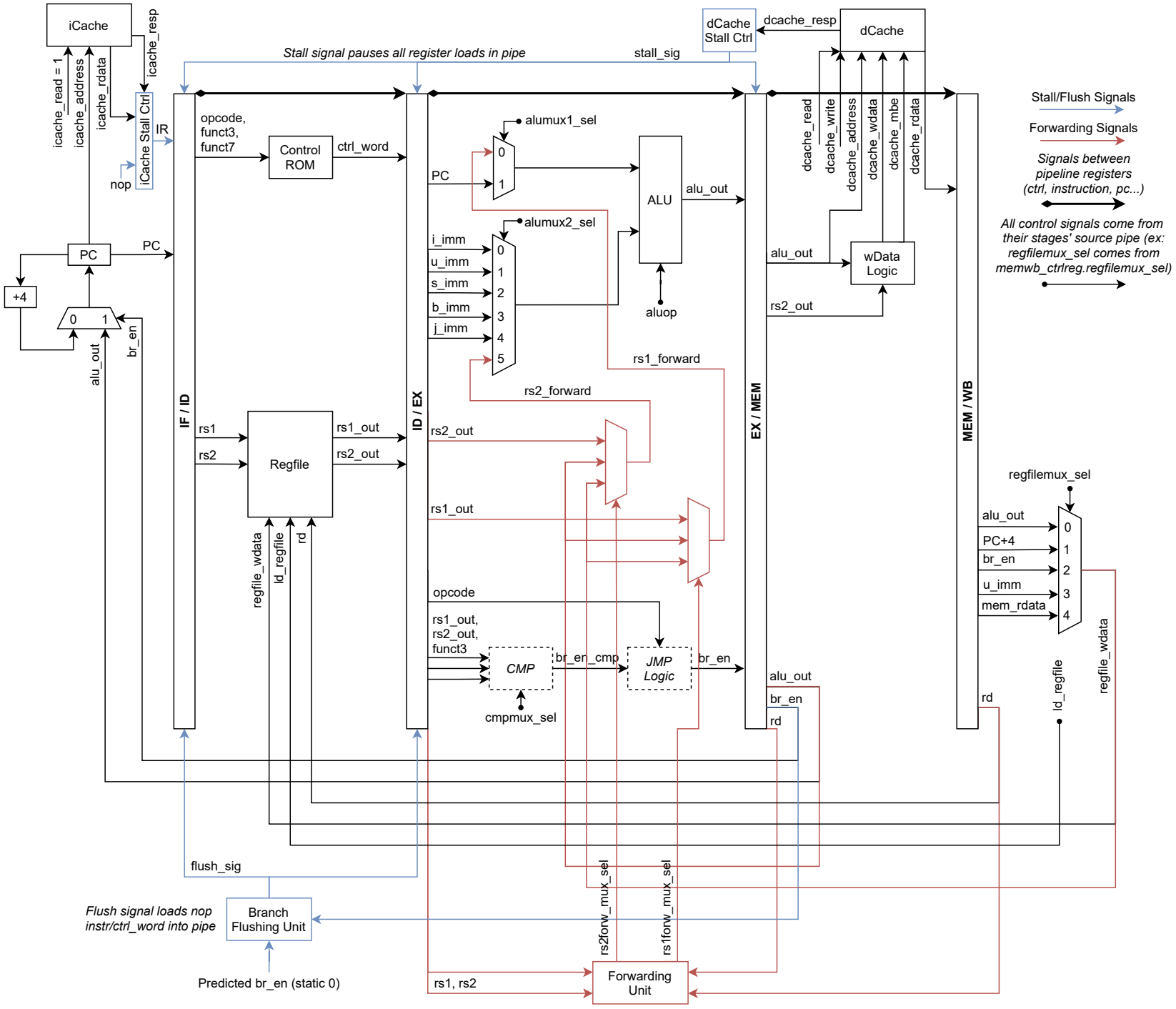
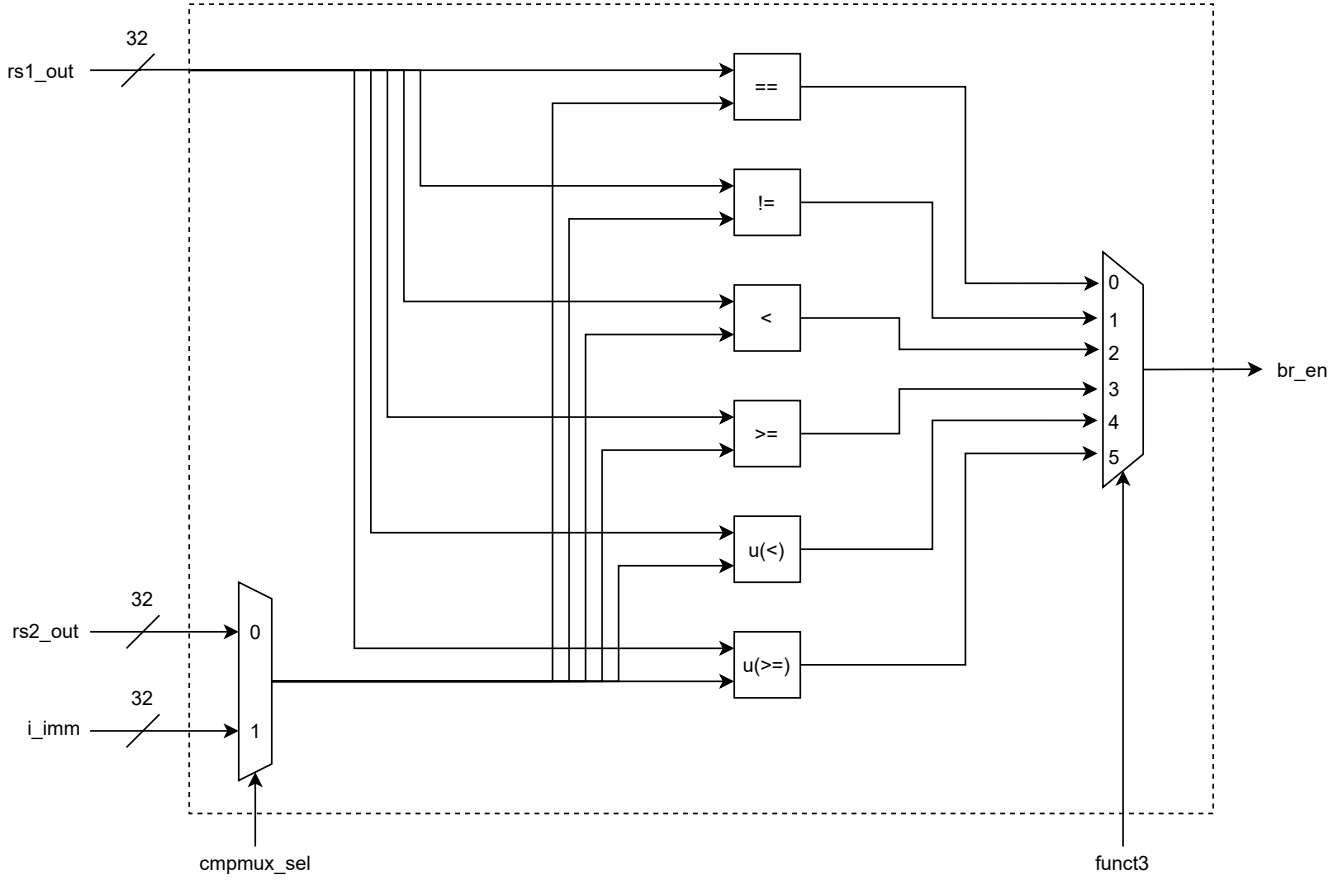
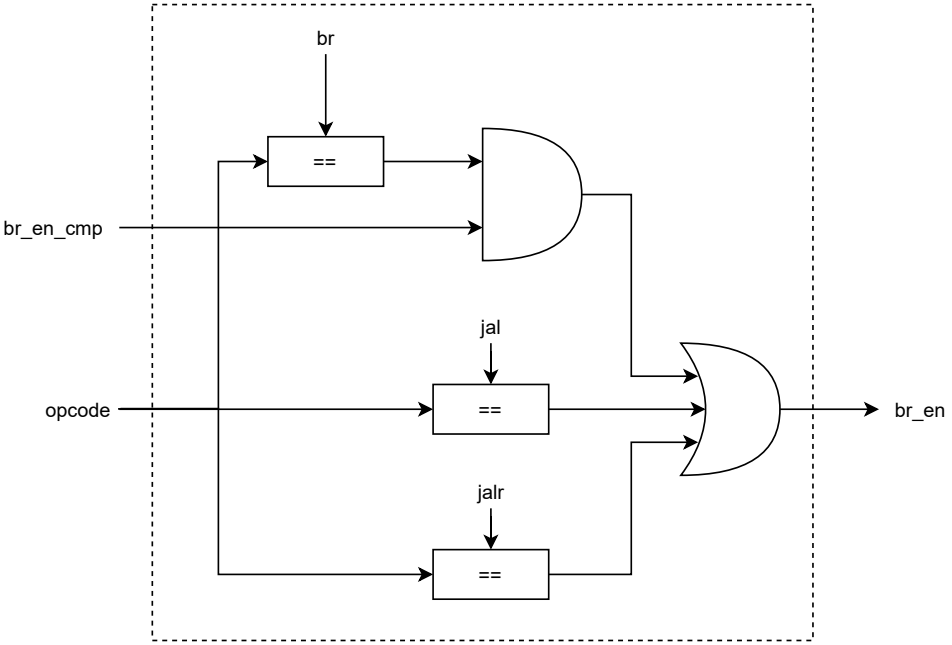

iCache
icache_read = 1
icache_address
icache_rdata
icache_resp
iCache Stall Ctrl
IR
nop
Stall signal pauses all register loads in pipe
stall_sig
dCache Stall Ctrl
dcache_resp
dCache
opcode, funct3, funct7
Control ROM
ctrl_word
alumux1_sel
PC
ALU
alu_out
alumux2_sel
i_imm
u_imm
s_imm
b_imm
j_imm
aluop
dcache_read
dcache_write
dcache_address
dcache_wdata
dcache_mbe
dcache_rdata
wData Logic
rs2_out
Stall/Flush Signals
Forwarding Signals
Signals between pipeline registers (ctrl, instruction, pc...)
All control signals come from their stages' source pipe (ex: regfilemux_sel comes from memwb_ctrlreg.regfilemux_sel)
+4
PC
br_en
rs1_forward
rs2_forward
IF / ID
ID / EX
EX / MEM
MEM / WB
rs1
rs2
Regfile
rs1_out
rs2_out
regfile_wdata
ld_regfile
rd
opcode
rs1_out, rs2_out, funct3
CMP
br_en_cmp
JMP Logic
cmpmux_sel
regfilemux_sel
PC+4
mem_rdata
flush_sig
Flush signal loads nop instr/ctrl_word into pipe
Branch Flushing Unit
Predicted br_en (static 0)
rs2forw_mux_sel
rs1forw_mux_sel
rs1, rs2
Forwarding Unit

CMP

Key:
u(<) : Unsigned less than
u(>=) : Unsigned greater than equal to



JMP Logic



Forwarding Unit Logic

Inputs:
idex_ireg_out.rs1
idex_ireg_out.rs2
exmem_ireg_out.rd
memwb_ireg_out.rd

Outputs:
rs1forw_mux_sel
rs2forw_mux_sel

Defaults:
rs1forw_mux_sel = forw_mux::idex_rs1reg_out
rs2forw_mux_sel = forw_mux::idex_rs1reg_out

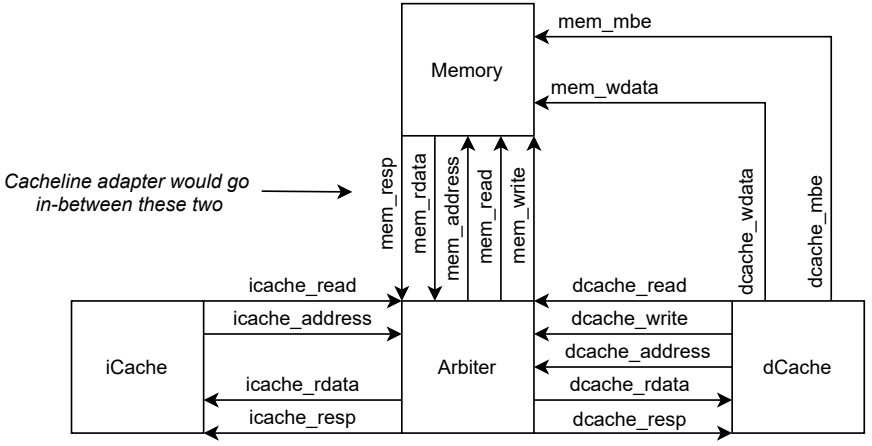
if exmem_ctrlreg_out.regfile_ld & (exmem_ireg_out.rd != 0)
& (exmem_ireg_out.rd = idex_ireg_out.rs1))
& (exmem_ireg_out.rd = idex_ireg_out.rs1))
    rs1forw_mux_sel = forw_mux::exmem_alureg_out

if exmem_ctrlreg_out.regfile_ld & (exmem_ireg_out.rd != 0)
& (exmem_ireg_out.rd = idex_ireg_out.rs2))
    rs2forw_mux_sel = forw_mux::exmem_alureg_out

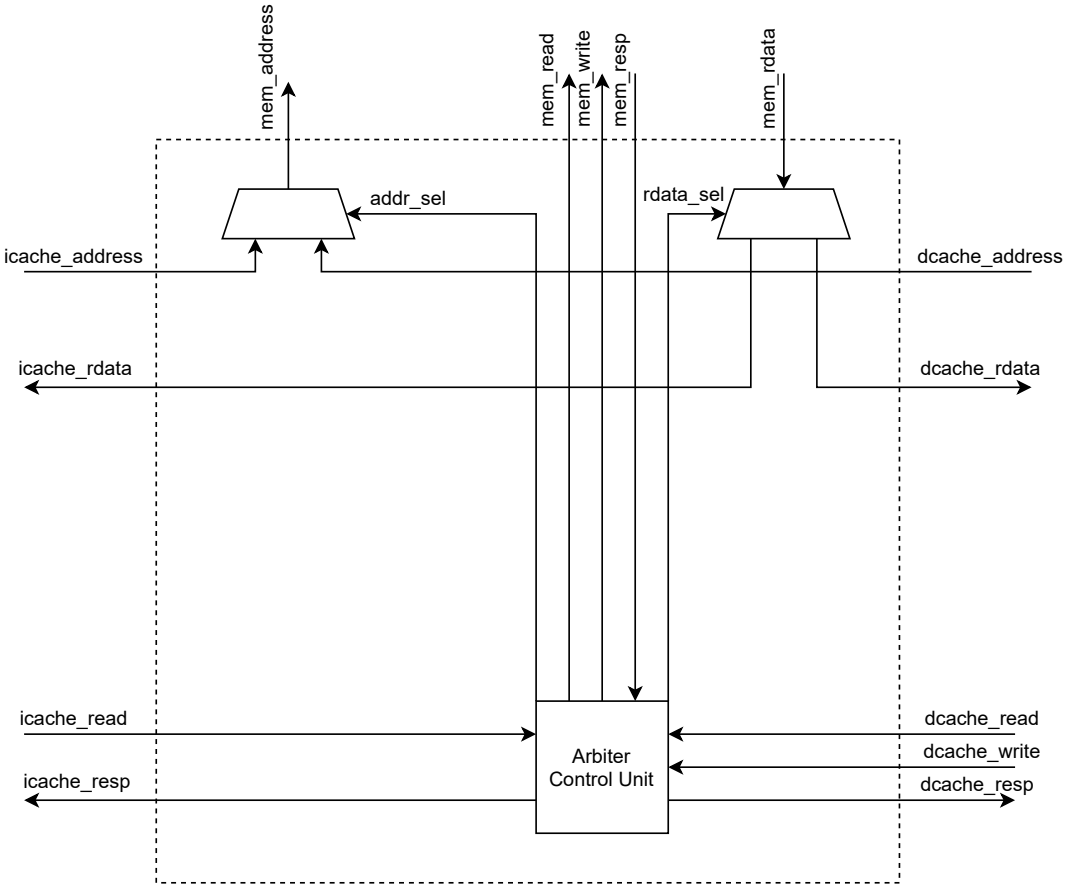
if (memwb_ctrlreg_out.regfile_ld & (memwb_ireg_out.rd != 0)
& !(exmem_ctrlreg_out.regfile_ld & (exmem_ireg_out.rd != 0)
& (exmem_ireg_out.rd != idex_ireg_out.rs1))
& (memwb_ireg_out.rd = idex_ireg_out.rs1))
    rs1forw_mux_sel = forw_mux::regfile_wdata

if (memwb_ctrlreg_out.regfile_ld & (memwb_ireg_out.rd != 0)
& !(exmem_ctrlreg_out.regfile_ld & (exmem_ireg_out.rd != 0)
& (exmem_ireg_out.rd != idex_ireg_out.rs1))
& (memwb_ireg_out.rd = idex_ireg_out.rs1))
    rs1forw_mux_sel = forw_mux::regfile_wdata

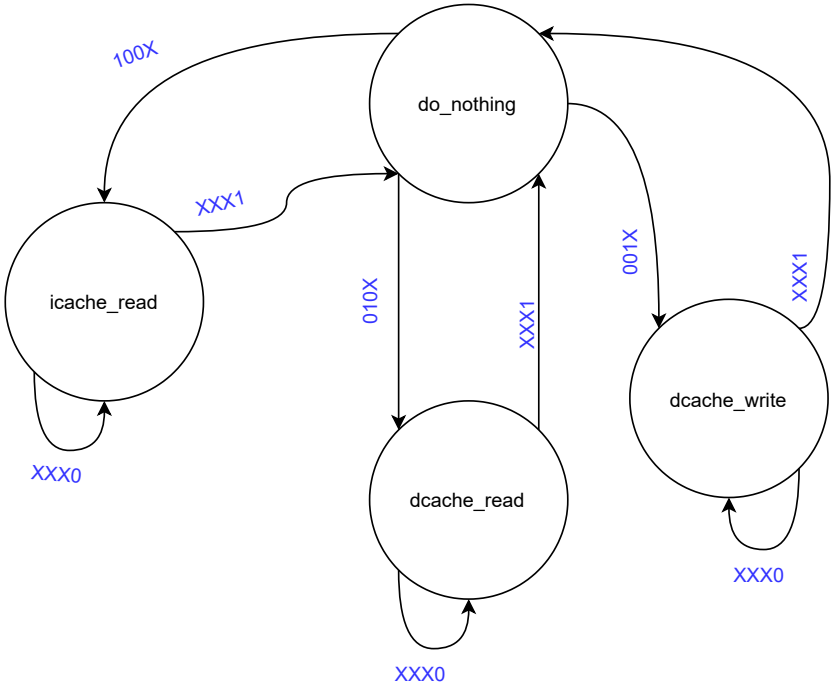
Memory Hierarchy



Arbiter


mem_address
mem_read
mem_write
mem_resp
mem_rdata
addr_sel
rdata_sel
icache_address
dcache_address
icache_rdata
dcache_rdata
icache_read
dcache_read
dcache_write
Arbiter
Control Unit
icache_resp
dcache_resp

Arbiter Control Unit State Machine



Bit representation in State Machine (L to R):
Bit 0: icache_read
Bit 1: dcache_read
Bit 2: dcache_write
Bit 3: mem_resp

Output at each state:

do_nothing
- mem_read = 0
- mem_write = 0
- addr_sel = addr_mux::icache_address
- rdata_sel = rdata_mux::icache_rdata
- icache_resp = 0
- dcache_resp = 0

icache_read
- mem_read = 1
- mem_write = 0
- addr_sel = addr_mux::icache_address
- rdata_sel = rdata_mux::icache_rdata
- icache_resp = mem_resp
- dcache_resp = 0

dcache_read
- mem_read = 1
- mem_write = 0
- addr_sel = addr_mux::dcache_address
- rdata_sel = rdata_mux::dcache_rdata
- dcache_resp = mem_resp
- icache_resp = 0

dcache_write
- mem_read = 0
- mem_write = 1
- addr_sel = addr_mux::dcache_address
- rdata_sel = rdata_mux::dcache_rdata
- dcache_resp = mem_resp
- icache_resp = 0