314513064-lab2-report

1. Introduction

We implement a 5-stage RV32I core that evolves from stall-only to bypassed and to a mul/div-pipelined variant, completing RV32I+CSRW and RV32M (incl. mulh/mulhu/mulhsu) while retaining load-use stalls. Correctness is verified with provided and custom M-extension tests, and ubmark cycles/IPC across riscvstall/byp/long quantify throughput–latency trade-offs under the Iron Law.

2. Design

Objective1:

- **Full RV32I control**: Completed decode/control for all I/R/U/UJ/S/SB forms (incl. JALR, 6 branches, SUB, logic, shifts, SLT{,U}), wiring PC/ALU/Mem/WB/CSR so the stall-only core runs end-to-end.
- **RV32M basics integrated**: Added MUL/DIV/DIVU/REM/REMU with proper req/resp handshakes, execute-stage mux (ALU vs MulDiv), correct result pick (low for mul/div, remainder path for rem), and stalls when not ready.
- **Control-flow done right**: All six branches + JALR drive pc_mux_sel; mispredict triggers clean squashes.
- **Subword memory correct**: Added LB/LH/LBU/LHU/SB/SH; set dmemreq_msg_len and dmemresp_mux_sel for correct sign/zero-extend.
- **Robust stalling & observability**: Unified RAW/handshake stalls (rs vs X/M/W, *_val/rdy) with response queues to prevent drops; kept CSRW-toggled num_cycles/num_inst counters at a consistent counting point and improved InstMsg for debugging.

Objective2:

- **Unified 8-op decode & flow control:** One decoder for

MUL/DIV/DIVU/REM/REMU/MULH/MULHSU/MULHU; is_mul_family routes to a single active sub-unit. Requests are gated; backpressure via muldivreq_rdy = mulreq_rdy && divreq_rdy.

- **High-half sign policy outside the core:** Multiplier kept **unsigned 32×32→64**; MULH/MULHSU use abs×abs plus a final two's-complement negate, latched by negate_result_q.
- **Multiplier refactor:** IntMulIterative now pure unsigned shift-add, 32-cycle FSM (IDLE→CALC→SIGN); signed handling lives in the wrapper.
- **Divider unchanged:** IntDivIterative retains baseline signed/unsigned handling and {rem, div} 64-bit return; my changes focus on multiply/top wrapper.
- **Race-free responses & result pick:** Top prefers mulresp, else divresp; gated requests prevent same-cycle dual valid. WB selects low/high 32 as required (MUL→low, MULH*→high, DIV→quot low, REM→rem high).

Objective3:

- **Real bypassing:** rs1/rs2 come from a bypass mux (X/M/W), not just RF → removes most ALU bubbles.
- **Tight hazard policy:** bypass-by-default; stall only for load-use, in-flight mul/div, or dmem handshake. Stalls are aggregated precisely (no broad freezes).
- **Control-flow aligned:** all 6 branches + JALR drive pc_mux_sel; taken branches squash upstream correctly.
- **Subword loads robust:** LB/LH/LBU/LHU/SB/SH with correct sign/zero-extend; 1-deep dmem response queue prevents drops.
- **M-ext & handshakes integrated:** added MULH/MULHSU/MULHU (take high-32); clean ALU⇄MulDiv execute mux, val/rdy wiring, and a unified X→M→W writeback path.
- **Metrics intact:** kept CSR counters and lightweight disassembly/PC hooks for apples-to-apples IPC and easy tracing.

Objective4:

Objective 4 targets integrating a long-latency (ideally pipelined) Mul/Div so the 5-stage core stalls only on true dependencies. In my riscvlong, I enabled MULH/MULHSU/MULHU with correct high/low-32 selection, connected Mul/Div at X with val/rdy so Execute stalls only while a response is pending, and retained the ALU↔MulDiv result mux with 1-deep imem/dmem skid buffers; stalls are aggregated at X (Mul/Div + memory) with M back-pressure. I did not replace the iterative unit with a pipelined one or extend forwarding, so functionality is correct but throughput overlap is not realized yet.

3. Testing Methodology

- **Unit → Integration:** Treat IntMulIterative as **unsigned 32×32→64** and run directed cases on {0,1,−1, INT_MIN/MAX}, powers-of-two, and mixed magnitudes; verify wrapper semantics (MUL→low32, MULH/MULHSU/MULHU→high32; MULH commutative, MULHSU non-commutative); then mix MUL*/ALU/LD-ST sequences and use **SRCEQ/BYP** sections to stress RAW under stall-only control.
- **Directed per-instruction:**
  MUL—zeros/ones/negatives, cross-boundary products, src/dst overlap and BYP variants.
  MULH—operand-swap symmetry; boundaries like (-2^31)^2 → hi=0x40000000, (2^16)^2 → hi=1.
  MULHSU—non-commutativity; extremals (-1)*UINT_MAX, (-2^31)*{1,2}, and bit-k probes (-1)*2^k.
- **Corner cases:** 0*x / 1*x / (-1)*x; **INT_MIN/MAX/UINT_MAX** with small factors & powers-of-two; commutativity rules (MUL/MULH yes, MULHSU no); RAW hazards & src/dst overlap via **SRCEQ/BYP**.
- **Procedure & acceptance:** Harness with TEST_RISCV_BEGIN/END and TEST_RR_OP + SRCEQ/BYP macros for auto-check/fail; runs via make check-asm-riscvstall and make check-asm-rand-riscvstall; optional debug flags +disasm=3 +stats=1 +vcd=1.

4. Evaluation

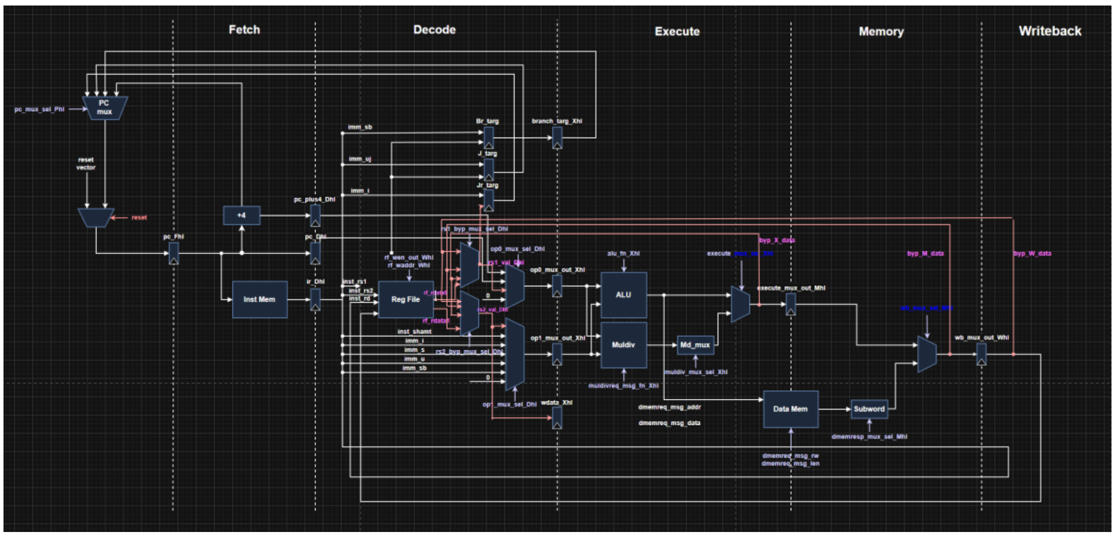
bench	core	cycles	instret	IPC	CPI
vvadd	stall	1778	1064	0.598	1.671
vvadd	byp	1447	1064	0.735	1.360
cmplx-mult	stall	18687	2949	0.158	6.337
cmplx-mult	byp	16685	2949	0.177	5.658
masked-filter	stall	21032	6964	0.331	3.020
masked-filter	byp	17701	6964	0.393	2.542
bin-search	stall	3258	1126	0.346	2.893
bin-search	byp	1539	1126	0.732	1.367

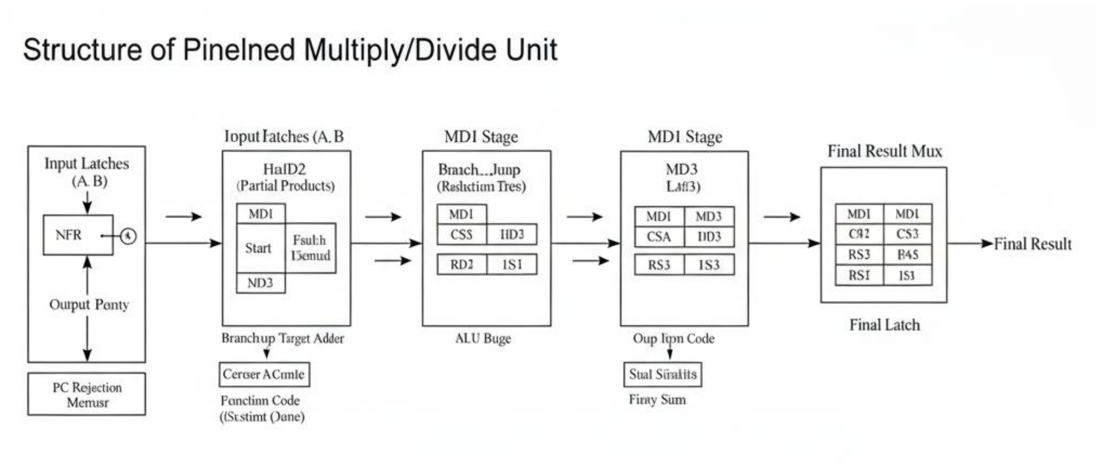
We evaluate four ubmarks (vvadd, cmplx-mult, masked-filter, bin-search) on the 5-stage core with and without bypassing. For each run, we record total cycles and retired instructions (+stats) and report IPC = instret / cycles.

5. Discussion:

This work compares only the five-stage processor configurations with and without bypassing. On the four ubmarks I measured, bypassing yields speedups of 1.23×/1.19×/1.12× for vvadd, masked-filter, and cmplx-mult, respectively, and 2.12× for bin-search; the IPC likewise increases from stall→bypass as 0.598→0.735, 0.331→0.393, 0.158→0.177, and 0.346→0.732. The primary reason is that bypassing removes the wait caused by producer→consumer ALU data dependencies, which is especially beneficial for tight loops like bin-search that follow a "compare → branch → next compare" pattern. However, load–use hazards can still introduce a one-cycle stall and control hazards are unaffected, so the IPC remains well below 1. A limitation of this study is that it does not evaluate the 7-stage pipeline or the pipelined mul/div unit; we expect riscvlong to reduce long-latency stalls from M-extension instructions while incurring a larger branch penalty.

6. Figures:


Fetch
Decode
Execute
Memory
Writeback
PC mux
pc_mux_sel_Phl
reset vector
reset
+4
pc_Fhl
pc_plus4_Dhl
pc_Dhl
ir_Dhl
Inst Mem
imm_sb
imm_uj
imm_i
Br_targ
J_targ
Jr_targ
branch_targ_Xhl
rf_wen_out_Whl
rf_waddr_Whl
inst_rs1
inst_rs2
inst_rd
Reg File
rf_rdata0
rf_rdata1
rs1_byp_mux_sel_Dhl
rs2_byp_mux_sel_Dhl
op0_mux_sel_Dhl
op1_mux_sel_Dhl
op0_mux_out_Xhl
op1_mux_out_Xhl
inst_shamt
imm_i
imm_s
imm_u
imm_sb
alu_fn_Xhl
ALU
Muldiv
muldivreq_msg_fn_Xhl
Md_mux
muldiv_mux_sel_Xhl
execute
execute_mux_out_Mhl
byp_X_data
byp_M_data
byp_W_data
wdata_Xhl
dmemreq_msg_addr
dmemreq_msg_data
Data Mem
dmemreq_msg_rw
dmemreq_msg_len
Subword
dmemresp_mux_sel_Mhl
wb_mux_out_Whl


Structure of Pinelned Multiply/Divide Unit
Input Latches (A.B)
NFR
Output Penty
PC Rejection Memusr
Ioput Iatches (A.B
HaID2 (Partial Products)
MD1
Start
Fsulh I5emud
ND3
Branchup Target Adder
Cerser ACmle
Fonctinn Code (ISustimt Oone)
MD1 Stage
Brach...Junp (Reshctim Tres)
MD1
CS3
IID3
RD2
1S1
ALU Buge
MD1 Stage
MD3 Laff3)
MD1
MD3
CSA
DD3
RS3
1S3
Oup Ion Code
Stal Smalits
Finy Sum
Final Result Mux
MD1
MD1
C42
CS3
RS3
B45
RS1
IS3
Final Latch
Final Result