

314513064-lab3-report

1. INTRODUCTION

We build a two-wide, in-order superscalar RISC-V core: Part 1 adds dual-fetch/single-issue; Part 2 enables dual-issue via heterogeneous A/B pipelines with steering and a scoreboard; we then compare riscvlong, riscvdualfetch, and riscvssc on ubmark (cycles/IPC) to assess when dual-issue helps.

2. DESIGN

2.1 Baseline and Pipeline Organization (7-stage, two-wide)

- The datapath contains two parallel pipelines under an in-order machine:
 - **Pipeline A** supports ALU, memory, control-flow, and M-extension (mul/div).
 - **Pipeline B** supports simple ALU operations only (no mul/div, no branch, no memory).

2.2 Part 1: Dual-Fetch, Single-Issue

- **Fetch and alternating single issue.** Each cycle the front-end fetches two instructions using `pc_plus8_*`. After decode, only **one** instruction is issued; the other remains in Decode and issues on the following cycle. On taken branches/jumps, the **second** instruction of the pair is **killed** to prevent re-execution on the wrong path.
- **Implementation notes (our design).**
 - Fixed a potential full-pipe stall caused by X1 (memory) responses: we made the **X0→X1 data-memory handshake two-state safe** so a request cannot drop when X0 is stalled (eliminates timeouts/X values).
 - PC/control interaction: verified the fetch stride (`pc_plus8_*`) and ensured that the second instruction in a fetch pair is flushed on taken control flow.

2.3 Part 2: Dual-Issue (Steering + Scoreboard)

- **Steering rules (key cases).**
 - **ALU / ALU** → issue #0 to **A**, #1 to **B**.
 - **ALU / non-ALU** → #0 to **B**, #1 to **A**.
 - **non-ALU / ALU** → #0 to **A**, #1 to **B**.
 - **non-ALU / non-ALU** → #0 to **A**, **stall #1 one cycle** and then send it to **A**.
 - If both instructions would write the **same destination register** (WAW), **stall** the second to avoid dual-write conflicts and simplify forwarding.
- **Scoreboard responsibilities.**
 - At Decode/Issue, the scoreboard marks **Pending destinations** and determines for each source whether it is **ready** or requires **forwarding** (and from which pipeline/stage).
 - It detects **intra-pair RAW** (0→1). If the second instruction depends on the first and cannot be safely co-issued, it is delayed.
 - Because all writes occur at the tail, the scoreboard focuses on **readiness/forwarding and stalls**, rather than modeling multi-cycle write-back conflicts.
- **Implementation notes (our design).**
 - X1 now stalls **only** while a load/store awaits data; bookkeeping for the data-response queue is local to X1, avoiding global lockups.
 - The **dmem request is latched** and cleared **only after acceptance** (dmemreq_rdy), preventing memreq*_val from going unknown.
 - The **second-issue gate** requires “B-capable + no WAW/Raw that forbids pairing + structural allowance,” aligning with the heterogeneous lanes.

3. TESTING METHODOLOGY

We validated the two-wide in-order riscvssc with dual-fetch, steering, and a scoreboard by running a directed assembly using `./riscvssc-sim +exe=../tests/build/vmh/riscv-test.vmh +disasm=3`. The program covers five cases: WAW on x5 (second writer delayed, final x5=30), independent non-ALU (`lui/auipc`, $x12=0x12345000$), RAW ALU chain (`add→sub`, $x18=20$), independent ALU pair (dual-issue to A/B, $x24=13$, $x25=4$), and WAW on x6 (serialized, $x6=50$). The run ends with *** PASSED ***. Micro-trace lines show true dual-issue, correct branch kills, and scoreboard-induced holds before safe release on RAW/WAW. No timeouts or X-propagation were observed; the `$readmemh` warning is benign. These results confirm correct steering, hazard handling, and effective dual-issue on independent pairs.

ID	Purpose	Pattern (abridged)	Expectation	Check
T1	WAW in same pair	<code>addi x5,...;</code> <code>addi x5,...</code>	2nd stalls; final x5=30	<code>bne x5,x11,_fail</code>
T2	Independent non-ALU	<code>lui x12,...;</code> <code>auipc x13,...</code>	Co-issue if allowed; correct regs	$x12==0x12345000$
T3	RAW on ALU	<code>add x17,...;</code> <code>sub x18,x17,...</code>	Stall or forward; x18==20	<code>bne x18,x19,_fail</code>
T4	Independent ALU pair	<code>add x24,...;</code> <code>sub x25,...</code>	Co-issue (0→A,1→B)	$x24==13$, $x25==4$

4. EVALUATION

bench	core	num_cycle	num_inst	ipc
Bin-search	riscvlong	1539	1126	0.731644
Bin-search	riscvlong(rand)	5584	1124	0.201289
Bin-search	dualfetch	2127	118	0.055477

Bin-search	dualfetch(rand)	9264	31	0.003346
Bin-search	riscvssc	160	65	0.406250
Bin-search	riscvssc(rand)	595	56	0.094118
Masked-filter	riscvlong	10784	6964	0.646772
Masked-filter	riscvlong(rand)	61225	6962	0.113712
Masked-filter	dualfetch	13625	694	0.050936
Masked-filter	dualfetch(rand)	54441	98	0.001800
Masked-filter	riscvssc	timeout	timeout	—
Masked-filter	riscvssc(rand)	timeout	timeout	—
cplx-mult	riscvlong	5635	2949	0.523336
cplx-mult	riscvlong(rand)	39834	2948	0.074007
cplx-mult	dualfetch	5169	17	0.003289
cplx-mult	dualfetch(rand)	36055	4	0.000111
cplx-mult	riscvssc	timeout	timeout	—
cplx-mult	riscvssc(rand)	timeout	timeout	—
vvadd	riscvlong	1447	1064	0.735314
vvadd	riscvlong(rand)	9786	1063	0.108625
vvadd	dualfetch	1961	88	0.044875
vvadd	dualfetch(rand)	13665	18	0.001317
vvadd	riscvssc	timeout	timeout	—
vvadd	riscvssc(rand)	timeout	timeout	—

Baseline > dual-fetch/dual-issue now (steering/hazard/DMEM limits; A-only bottleneck). With counter fix, 1-cycle non-ALU deferral, and stronger DMEM handshake, dual-issue should win on ALU-rich workloads and break even on memory/branch-heavy ones.

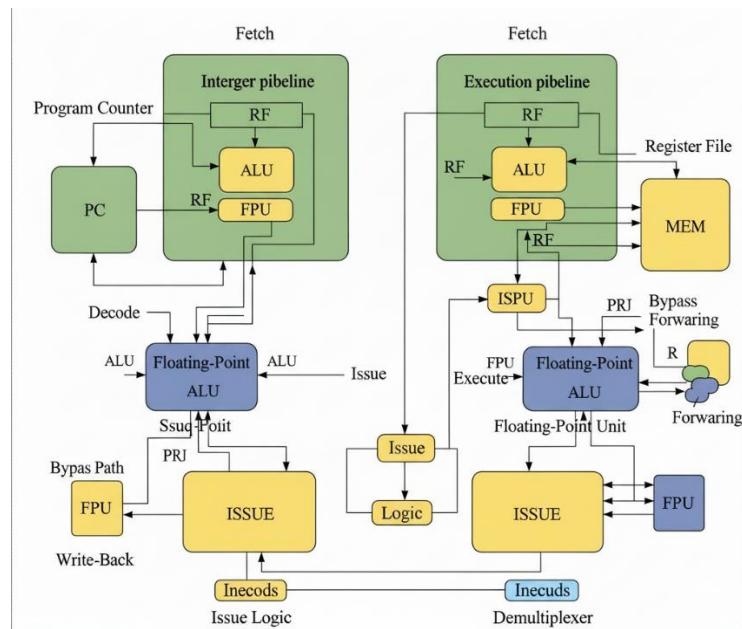


Figure1. Datapath diagram of heterogeneous dual-issue superscalar I4 processor

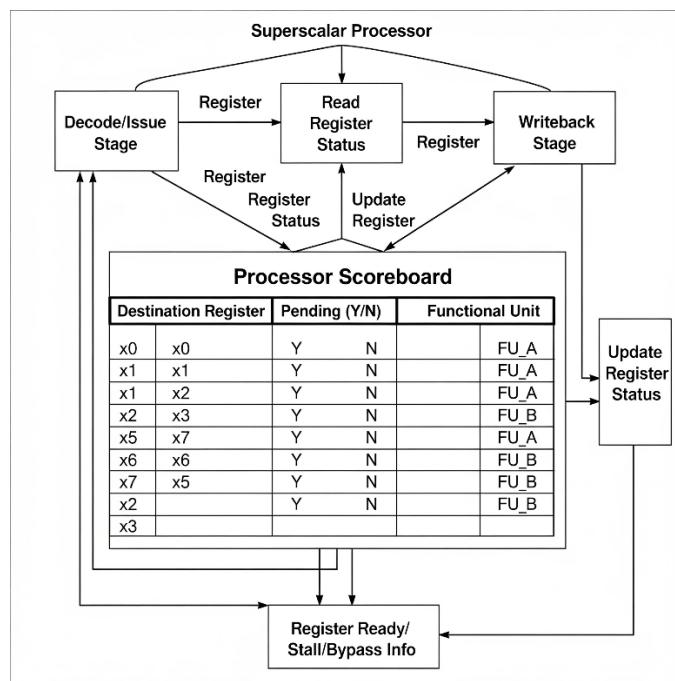


Figure2. Diagram of the implemented scoreboard