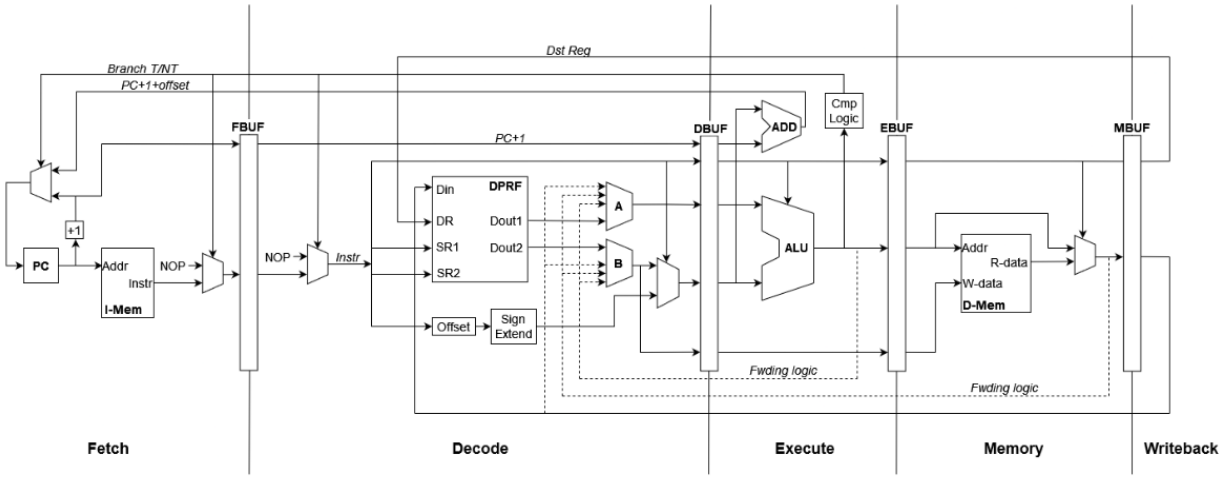
Network & Systems Pipelining Project Report

Author: Jinseo Lee

Collaborators: Minhong Cho, Taeho Kim

Overview

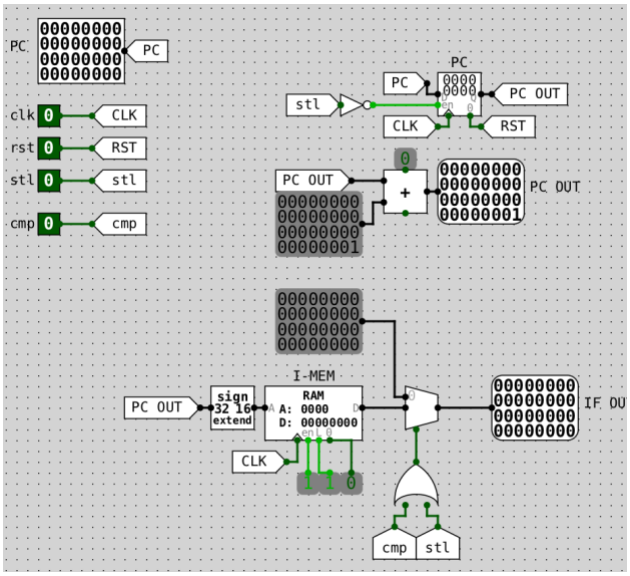


1. IF - Instruction Fetch
2. ID/RR - Instruction Decode/Register Read
3. EX - Execute (ALU operations)
4. MEM - Memory (both reads and writes with memory)
5. WB - Writeback (writing to registers)

Microcode & Instruction Loading

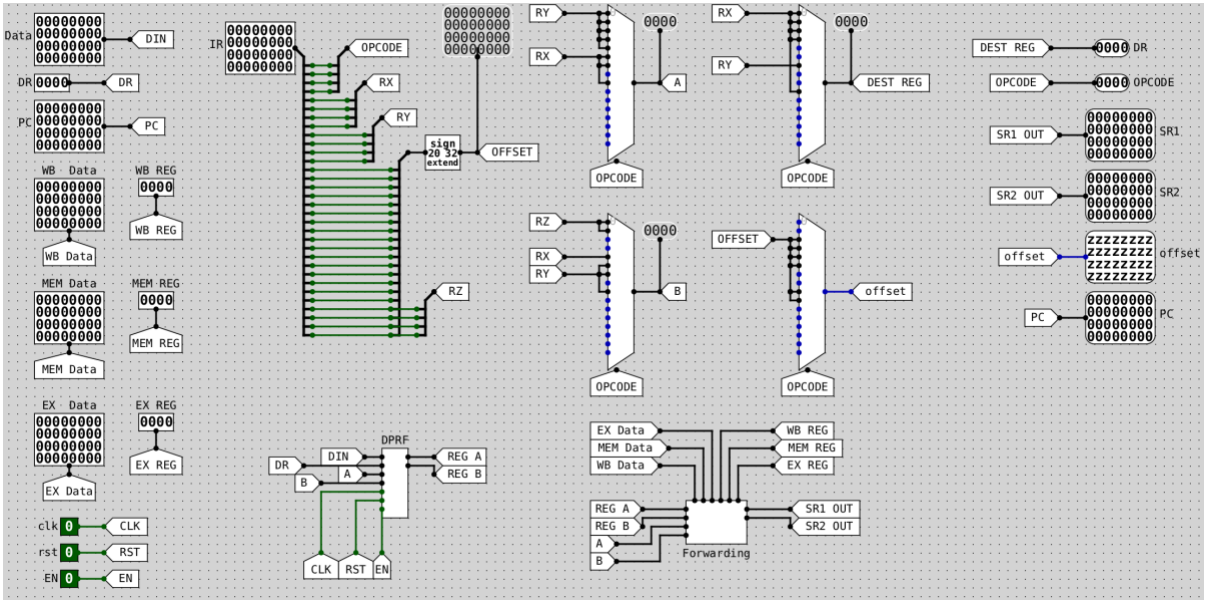
Instead of microcoded control, our pipeline is fed the assembler's machine code directly. Once we assemble each program (e.g. pow.s), we translate its 32-bit instructions into hexadecimal and store them in Instruction Memory (I-MEM) and D-MEM. This obviates any control-ROM: all control signals - opcode decoding, ALU function selection, branch evaluation, and memory enables - are provided by fixed combinational logic and minuscule opcode-driven multiplexers. Taking hex-converted instructions backfeeds a one-to-one correspondence between the ISA and the pipeline's control path.

IF



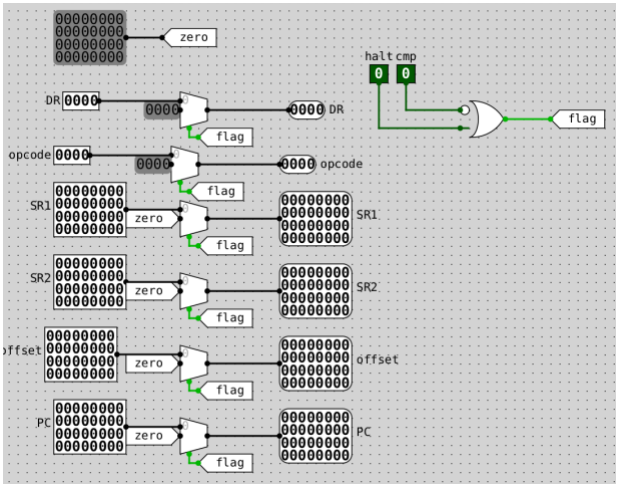
During the IF stage, the 32-bit program counter (PC) is first truncated to 16 bits via CircuitSim's built-in extender and used to address instruction memory (I-MEM). On each rising clock edge—unless a HALT has been detected—a two-input multiplexer chooses between PC + 1 for sequential execution and PC + offset for control transfers, where the offset is the sign-extended immediate calculated by the EX-stage. A dedicated comparison subcircuit evaluates BEQ and BGT conditions by feeding the ALU's subtractor output into equality and greater-than detectors, and an opcode check unconditionally triggers JALR jumps; their combined output forms a single branch flag. Simultaneously, a HALT decoder watches the opcode and, when HALT is asserted, disables the PC register's clock enable and forces the PC MUX's output to zero, preventing further instruction fetches. Finally, the newly chosen PC value and the fetched 32-bit instruction are latched into the IF/ID buffer (FBUF) on the same cycle, delivering stable, synchronized inputs to the next stage without incurring additional combinational delay.

ID/RR



In the ID/RR stage, FBUF presents the latched instruction and PC to a splitter network that parses the 32-bit word into its constituent fields: a 4-bit opcode, a 4-bit destination register (DR), two 4-bit source registers (RX, RY), and a 20-bit immediate/offset (sign-extended to 32 bits). These fields feed a dual-ported register file (DPRF), with RX and RY read onto operand buses A and B, while write-back data from the WB stage - gated by a global write-enable (WB-Enable AND Not-HALT) - writes into DR. To eliminate stalls due to data hazards, a forwarding subcircuit compares RX and RY against DR values residing in the EX, MEM, and WB pipeline registers. A three-input priority encoder (favoring EX, then MEM, then WB results) steers the most recent data onto the operand buses. On the rising edge, the ID/EX pipeline register (DBUF) captures the opcode, DR, forwarded operands, offset, and PC, ensuring that the EX-stage always receiwves hazard-free inputs.

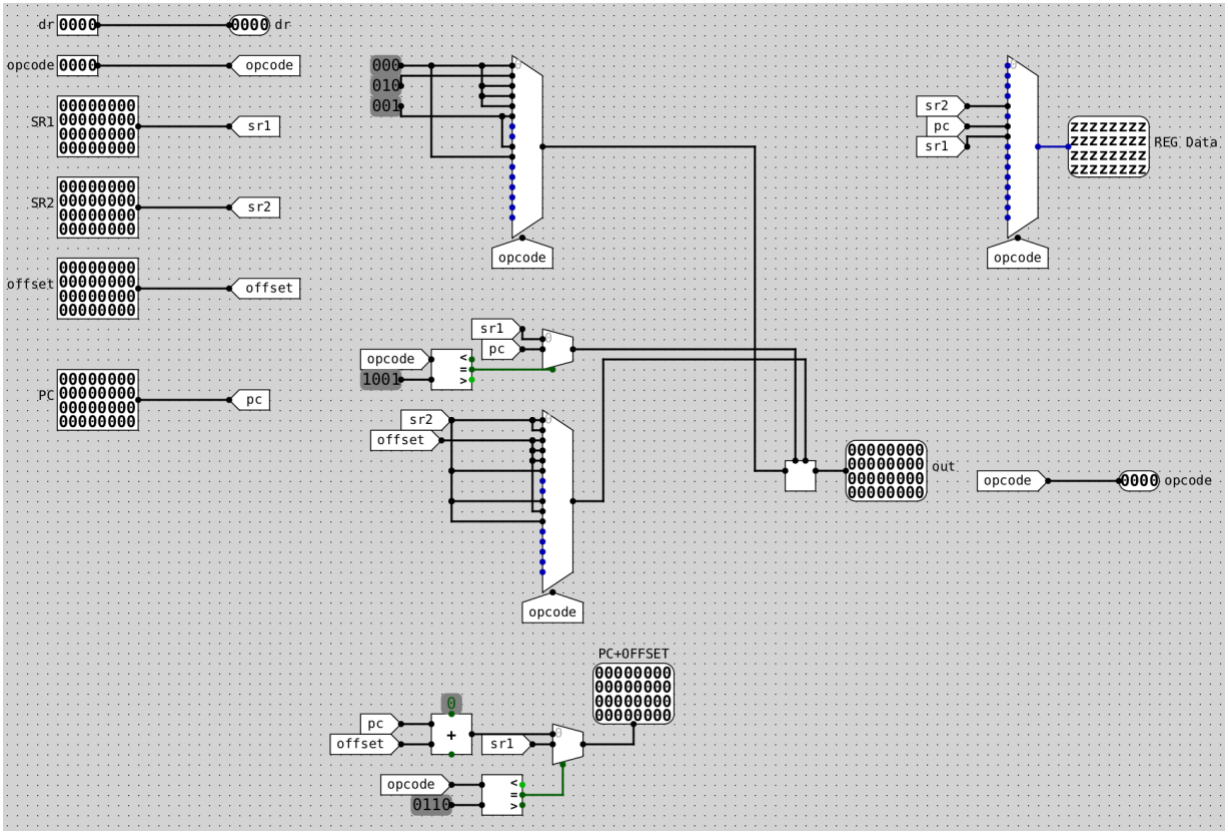
Flush



Immediately after ID/RR, our Flush subcircuit handles control hazards by using a single “flush” flag to steer six two-input multiplexers - one each for DR, opcode, SR1, SR2, offset, and PC.

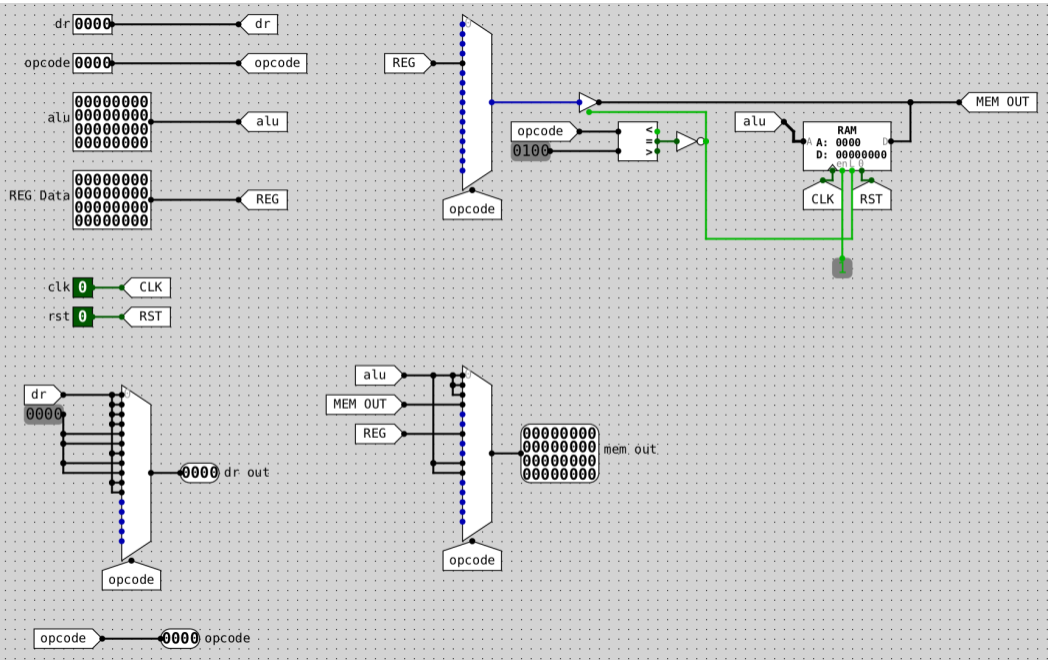
When flush = 0 (no branch taken), each MUX simply forwards its ID/RR value into DBUF on the clock edge. The moment the branch comparator or JALR detector asserts flush = 1, all six MUXes switch to a constant zero vector, converting the in-flight instruction into a NOP. Because both the zero constants and the ID/RR outputs share the same bus lines, the MUXes prevent any bus contention or short circuits by allowing only a single driver at a time. On the following clock, DBUF latches these zeros (and the true PC for branch alignment), so the EX-stage executes a harmless NOOP while IF has already resumed fetching from the updated PC. This zero-injection flush mechanism recovers from control hazards without stalling the entire pipeline.

EX



Once DBUF presents its outputs, the EX-stage completes arithmetic, logic, and branch-target computations in one cycle using pure combinational logic. Two opcode-controlled multiplexers select ALU inputs: MUX A chooses between SR1 and PC (latter for LEA and JALR) while MUX B chooses between SR2 and the sign-extended offset (latter for ADDI, LW, SW, BEQ, and BGT). A third “func” MUX, keyed to the instruction’s low-order func bits, activates one of three subcircuits - adder, subtractor, or NAND gate - to perform the core operation. For branches, the subtractor’s output also drives two comparators (zero for BEQ, greater-than for BGT), and their OR (plus an unconditional JALR detect) forms the global branch flag. Concurrently, a dedicated adder computes PC + offset and supplies this branch target address to the IF stage whenever the flag is asserted. On the same clock edge, the EX/MEM buffer (EBUF) latches the ALU result or branch target, the second source operand (for SW), DR, and opcode. Because all control-flow and arithmetic operations are hard-wired from the opcode, no microcode ROM is required.

MEM



In the MEM stage, EBUF outputs feed a small combinational circuit that masks out the upper 16 bits of the ALU result, using only the lower 16 bits to address the LC-3200's 16-bit Data RAM. An opcode-controlled enable line gates writes only on store instructions (SW), while on load instructions (LW) the RAM's 32-bit output appears on the next clock. A 2:1 multiplexer, steered by opcode, selects between the RAM read data (for loads) and the original ALU result (for arithmetic and address computations). On the rising edge, the MEM/WB buffer (MBUF) captures this "MEM Data," DR, and opcode, delivering clear, correctly typed data to the WB stage without ambiguity between memory reads and ALU operations.

WB

Finally, in the WB stage, MBUF outputs drive a last write-back multiplexer: for LW it selects MEM Data; for all other writing instructions (ADD, NAND, ADDI, LEA, JALR), it selects the ALU result. The chosen 32-bit value and DR feed the write port of the DPRF, with a global write-enable asserted for any register-writing instruction and gated by NOT HALT. A simple hardware check prevents writes to register 0, preserving its constant-zero semantics. This completes our fully pipelined, five-stage data path: each instruction's result returns to the register file every cycle with minimal control logic and zero reliance on microcode ROMs.

Challenges & Results

The size of our pipeline alone - seventeen subcircuits interconnected and five tightly coupled stages of pipeline - was an intimidating debugging task. Interleaving the control and data paths meant that a single mis-wired signal could propagate glitches to many stages, making root-cause analysis extremely time-consuming. Specifically, it took more than two weeks of rigorous signal tracing, frequent MUX input isolation, and sanity checking of the branch comparator and HALT logic in order to catch a short circuit in the flush/DBUF interconnect. We also struggled with tiny forwarding-priority edge cases - verifying EX-stage results take precedence over MEM/WB when needed without causing combinational loops - and timing flush properly so that NOOP injection didn't contaminate branch targets. The two-ported register file needed to be gated with caution with write-enable and HALT signals to avoid unintended writes to $zero or on pipeline stalls. Lastly, combining the LC-3200's unconditional JALR jump with BEQ/BGT flush control needed a special comparator that could resolve both unconditional and conditional transfers in a single cycle.

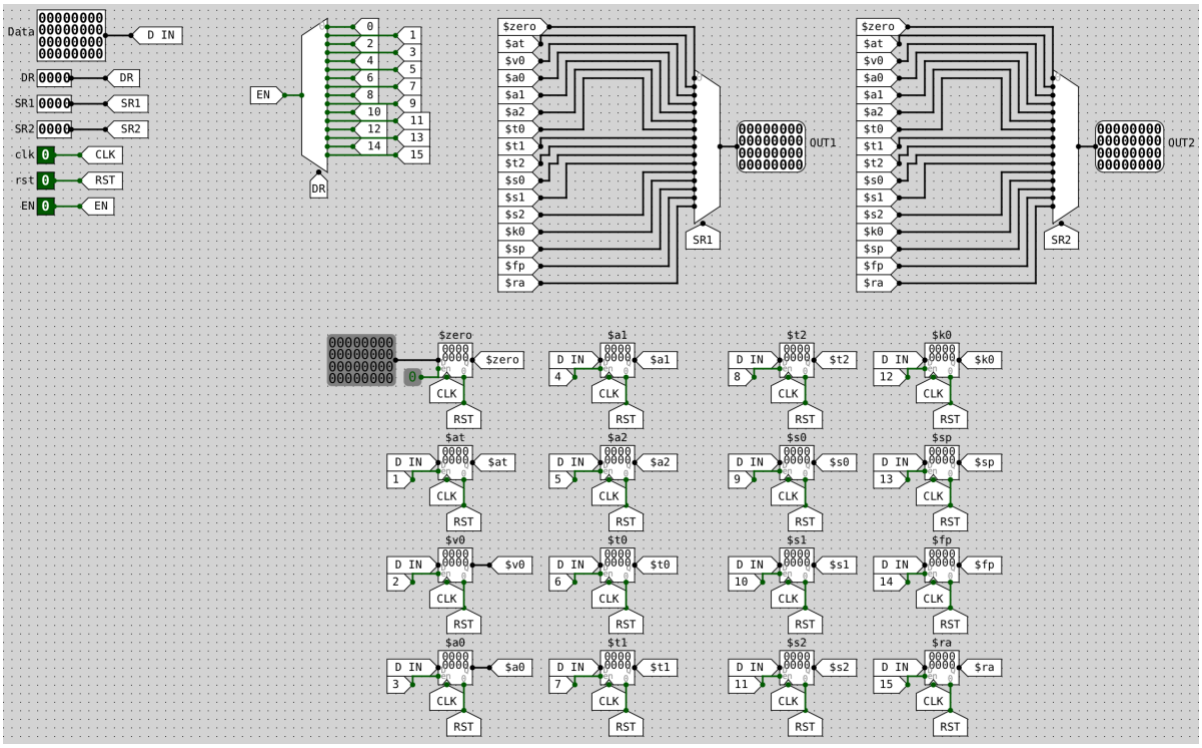
Although we could not manage a trustworthy cycle-count; counter would always report numbers way short of theoretical pipelined totals, we double-checked every instruction for accuracy scrupulously. We tested a whole series of hand-coded assembly tests, including the included pow.s benchmark, and in each instance register-file ending points were as expected. Load/store sequences, branch stalls, JALR returns, and even edge cases such as back-to-back data hazards all were dealt with perfectly by our flush and forwarding logic. Although real throughput figures are impossible to achieve due to tool-chain issues, functional correctness of our five-stage processor is completely proven.

Potential Areas of Improvement

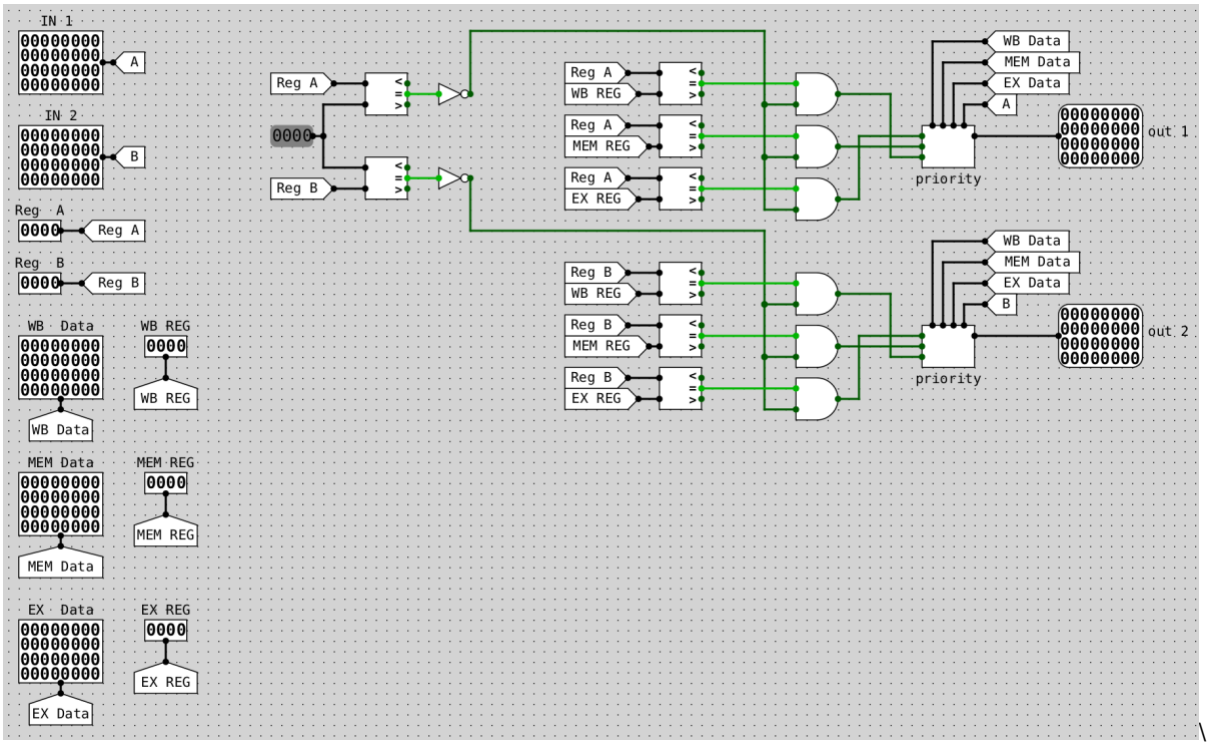
Although our design is functionally accurate, a few optimizations would boost performance and resilience. First, inclusion of a straightforward one-bit branch predictor (e.g. "predict-not-taken" with small branch history table) would lower pipeline flush rate on backward branches, boosting average IPC. Second, inclusion of special "load-to-use" stall detection in ID/RR would remove the single-cycle bubble still needed after a load, allowing fully hazard-free execution in most data-dependency cases. Third, pipelining or retiming the longest combinational paths - most notably the flush-MUX bank and branch-target adder - can reduce the cycle's critical path and enable a faster clock. Fourth, adding control logic to a microcode ROM can simplify the intricate web of opcode-driven multiplexers and comparators to be easier to extend in the future (e.g., new instructions) and be more gate-efficient. Finally, inserting a tiny instruction or data cache between the IF/ID and MEM phases would reduce mean memory latency substantially and provide a more accurate cycle-count estimate for programs such as pow.s.

Appendix

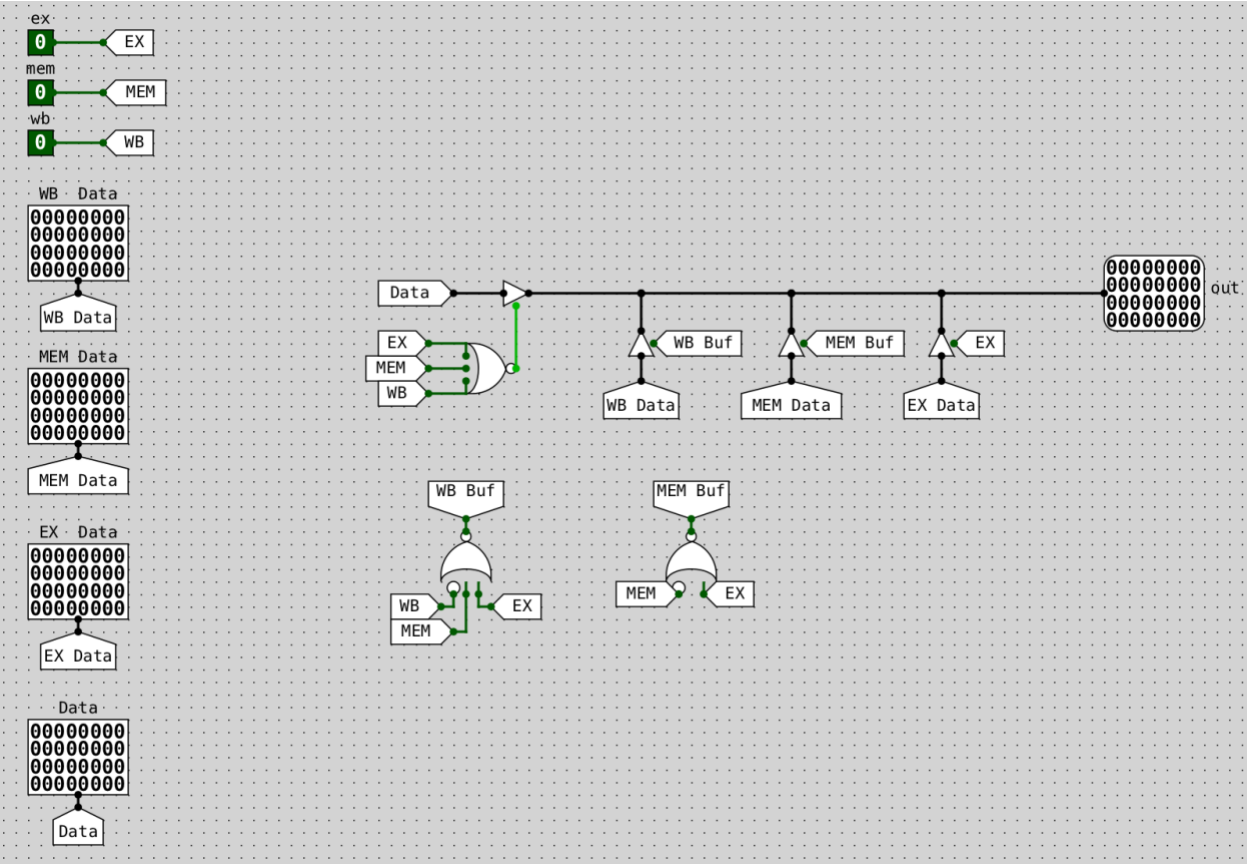
DPRF (Dual-Ported Register File)



Forwarding Logic



Priority



End of the Report.