Issue fixed by junior engineer

What have been observed:

- Arithmetic test vectors: `ADD` appears to perform subtraction (A gets smaller instead of larger).
- When testbench issues IN/MOV to register C, `C` stays at reset/old value.
- When testbench executes `OUT A`, `data_out` prints `zzzzzzzz` (high-Z) instead of A's value.

Use GTKWave to inspect these signals: `A`, `B`, `C`, `bus`, `LD`, `OE`, `InA`, `add_sub_out`, `data_out`, `inst`, `clk`.

1) Debug & fix Bug I — Arithmetic inversion in `add_sub` module

How to spot it

- In waveforms: after executing an `ADD R` instruction you expect `InA` / `add_sub_out` = `A + R`. Instead the waveform shows `A - R`.
- Alternatively, isolate ALU behavior with a small ALU unit test.

Quick ALU isolation testbench (optional but helpful)

Create `tb/add_sub_tb.v` to test ALU quickly (A=0x0F, bus=0x02):

```
module add_sub_tb();
  reg inst5;
  reg [7:0] A, bus;
  wire [7:0] out;
  add_sub dut (inst5, bus, A, out);
  initial begin
    A=8'h0F; bus=8'h02;
    inst5 = 0; #5 $display("inst5=0 ADD out=%h (expect 11)", out);
    inst5 = 1; #5 $display("inst5=1 SUB out=%h (expect 0D)", out);
    $finish;
  end
```

```
endmodule
```

Run: If results are swapped (inst5=0 => 0x0D, inst5=1 => 0x11) you have the inversion.

Root cause

In `add_sub module` the `assign` is reversed:

```
// BAD (injected bug)
assign add_sub_out = inst_5 ? (A + bus) : (A - bus);
```

Fix (exact edit)

Replace with:

```
// GOOD (fixed)
assign add_sub_out = inst_5 ? (A - bus) : (A + bus);
```

Why this fixes it: per spec `inst[5] == 1` selects SUB (A - bus). This mapping restores correct arithmetic.

Validate ALU fix

- Re-run the ALU unit test above. Expect:
  - `inst5=0` => out = `0x11` (A + bus)
  - `inst5=1` => out = `0x0D` (A - bus)

- Re-run full `micro_tb.v` and check that instructions performing ADD now increase A as expected.

2) Debug & fix Bug II — Register C load missing

How to spot it

- In the full simulation waveform: `LD[3]` (load enable for register C) toggles when you expect an IN/MOV to C; however `C` stays unchanged.

- For verification, after an IN C sequence, try `MOV C,A` or `OUT C` (use instructions that read C back to outside) — output not reflecting the new value.

Root cause

`bus_to_c` mux + `dff dC` lines were removed/commented out. No path writes the bus into C when `LD[3]` asserts.

Fix (exact edit)

Restore the missing lines in `micro.v`:

Add back:

```
// restore C load path
mux8_2to1 bus_to_c (LD[3], bus, C, C1);
dff dC (clk, C1, C);
```

Why this fixes it: When `LD[3]` is asserted, the mux selects `bus` as the next value for C (`C1`), and `dff` registers `C1` into `C` on the rising edge.

Validate C load fix

1. Re-run full testbench and inspect waveform:
   - At the cycle LD[3]==1, `C1` should equal `bus`, and at the next posedge, `C` should update to `C1`.
2. Add a TB assertion (optional) to automatically check C loading (see automation below).
3. Also verify downstream instructions that use C (e.g., ADD C) behave as intended.

3) Debug & fix Bug III — Tri-state buffer for A swapped (bus driven Z)

How to spot it

- In waveform: `OE[1]` is asserted during an `OUT A`, but `bus` remains `zzzzzzzz` and `data_out` shows Z.
- Confirm that other `from_X` drivers (for B, D, etc.) drive correctly on their OE signals.

Root cause

`from_A` tri-buffer was wired to a literal `8'bz` (or otherwise miswired) instead of `A`:

```
// BAD (injected)
tribuff_8_2to1 from_A (OE[1], 8'bz, bus);
```

Fix (exact edit)

Restore connection to `A`:

```
// GOOD (fixed)
tribuff_8_2to1 from_A (OE[1], A, bus);
```

Why this fixes it: Tri-state buffer now drives the bus with register A's value when `OE[1]` is true. Previously no driver existed for that OE, causing Z.

Validate tri-state fix

- Re-run full testbench.
- On `OUT A` cycles: `bus` and `data_out` should show A's value (not Z).
- Ensure there is no bus contention: only one OE should be asserted at a time — check `OE` signals in waveform.

4) Regression testing (after all fixes)

Re-run the full testbench

Check these:

- `ADD` and `SUB` produce the expected arithmetic results (inspect `A` and `add_sub_out`).
- `C` loads at the cycle after `LD[3]` asserted.
- `OUT A` drives `data_out` correctly.
- No unexpected `Z` on bus during normal operations.