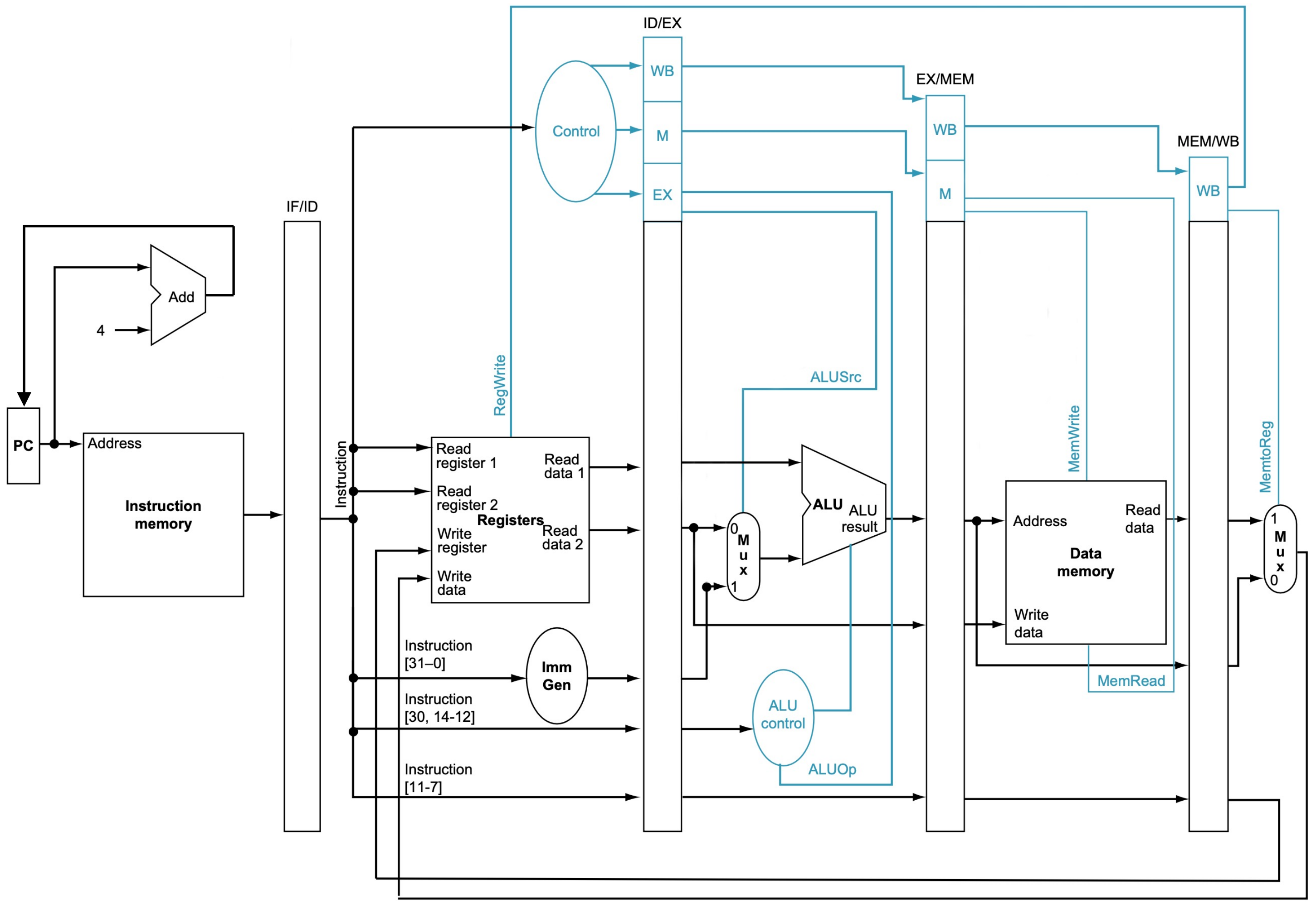
Lab 4-1 – Pipelined CPU
(w/o control flow instrs.)

CSED 311

Sungjun Cho

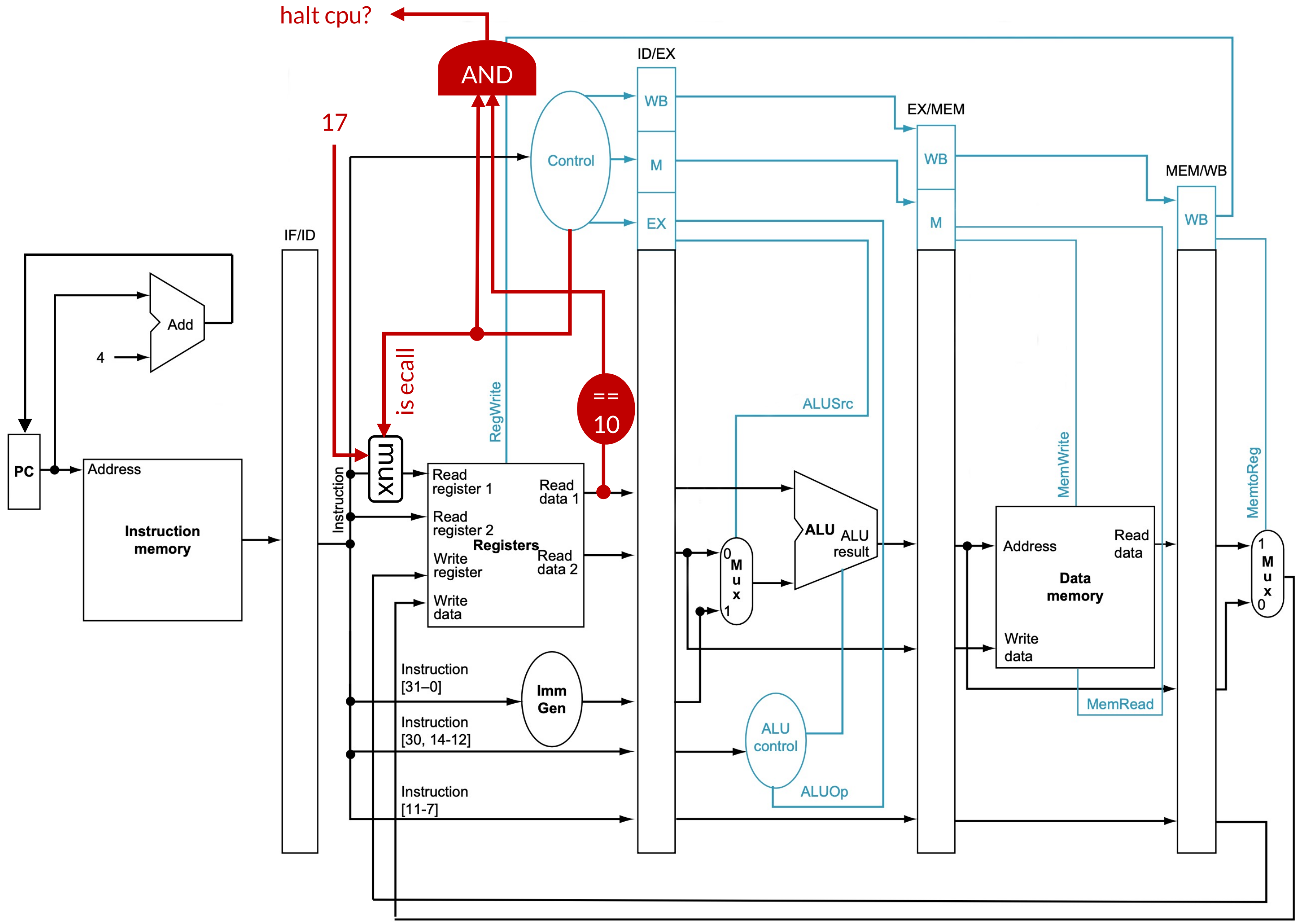
Datapath (w/o contol flow instrs.)

- You don’t have to implement control flow instructions now
  - E.g., JAL, BEQ, …



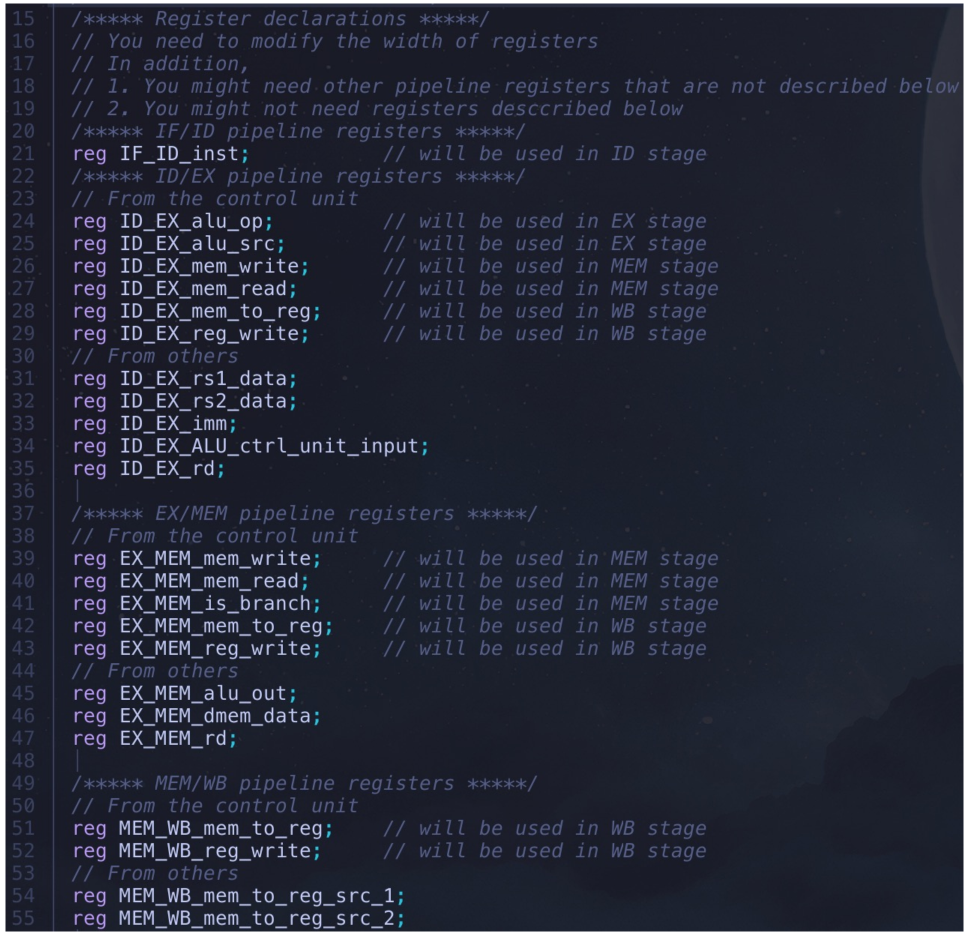
Datapath (w/o control flow instrs.)

- How to halt CPU?

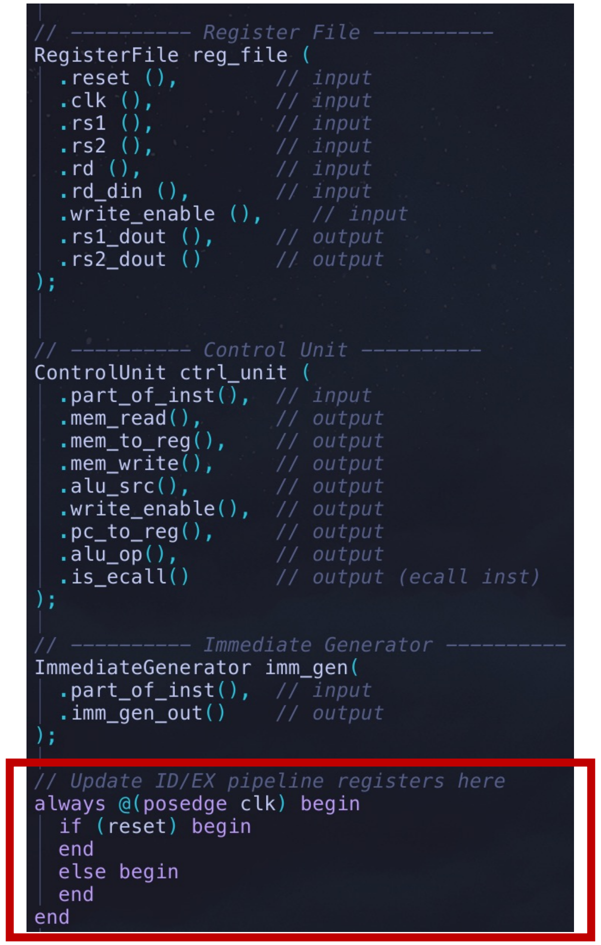


Update pipeline registers

- You need to understand how pipeline registers work
    - Pipeline registers are updated at rising edge of the clock



```verilog
/***** Register declarations *****/
// You need to modify the width of registers
// In addition,
// 1. You might need other pipeline registers that are not described below
// 2. You might not need registers desccribed below
/***** IF/ID pipeline registers *****/
reg IF_ID_inst;           // will be used in ID stage
/***** ID/EX pipeline registers *****/
// From the control unit
reg ID_EX_alu_op;         // will be used in EX stage
reg ID_EX_alu_src;        // will be used in EX stage
reg ID_EX_mem_write;      // will be used in MEM stage
reg ID_EX_mem_read;       // will be used in MEM stage
reg ID_EX_mem_to_reg;     // will be used in WB stage
reg ID_EX_reg_write;      // will be used in WB stage
// From others
reg ID_EX_rs1_data;
reg ID_EX_rs2_data;
reg ID_EX_imm;
reg ID_EX_ALU_ctrl_unit_input;
reg ID_EX_rd;

/***** EX/MEM pipeline registers *****/
// From the control unit
reg EX_MEM_mem_write;     // will be used in MEM stage
reg EX_MEM_mem_read;      // will be used in MEM stage
reg EX_MEM_is_branch;     // will be used in MEM stage
reg EX_MEM_mem_to_reg;    // will be used in WB stage
reg EX_MEM_reg_write;     // will be used in WB stage
// From others
reg EX_MEM_alu_out;
reg EX_MEM_dmem_data;
reg EX_MEM_rd;

/***** MEM/WB pipeline registers *****/
// From the control unit
reg MEM_WB_mem_to_reg;    // will be used in WB stage
reg MEM_WB_reg_write;     // will be used in WB stage
// From others
reg MEM_WB_mem_to_reg_src_1;
reg MEM_WB_mem_to_reg_src_2;
```



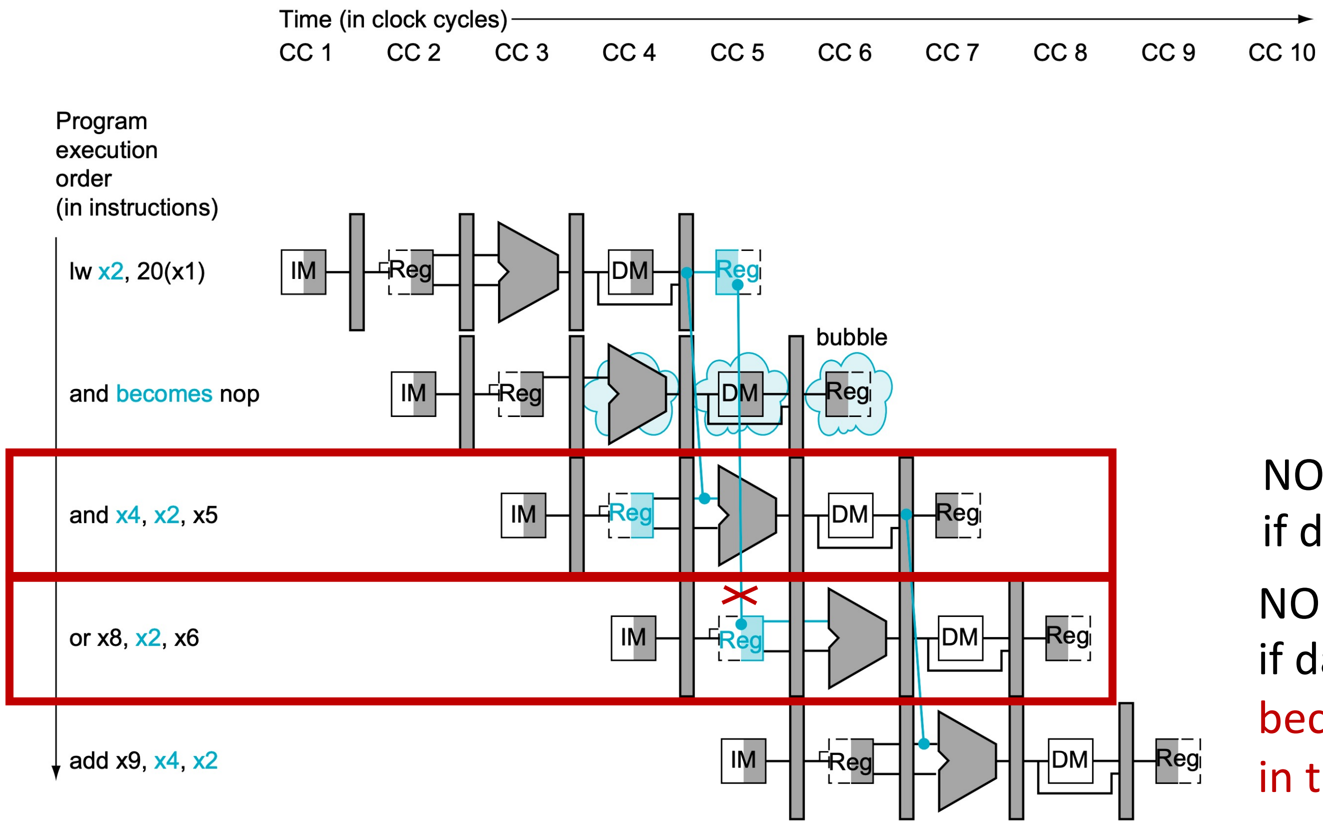
```verilog
// ---------- Register File ----------
RegisterFile reg_file (
  .reset (),          // input
  .clk (),            // input
  .rs1 (),            // input
  .rs2 (),            // input
  .rd (),             // input
  .rd_din (),         // input
  .write_enable (),     // input
  .rs1_dout (),       // output
  .rs2_dout ()        // output
);


// ---------- Control Unit ----------
ControlUnit ctrl_unit (
  .part_of_inst(),  // input
  .mem_read(),      // output
  .mem_to_reg(),    // output
  .mem_write(),     // output
  .alu_src(),       // output
  .write_enable(),  // output
  .pc_to_reg(),     // output
  .alu_op(),        // output
  .is_ecall()       // output (ecall inst)
);

// ---------- Immediate Generator ----------
ImmediateGenerator imm_gen(
  .part_of_inst(),  // input
  .imm_gen_out()    // output
);

// Update ID/EX pipeline registers here
always @(posedge clk) begin
  if (reset) begin
  end
  else begin
  end
end
```

Hazard

- Your implementation should resolve:
  - Data hazard
  - ~~Structural hazard~~
    - We do not append hardware modules to resolve structural hazard
  - ~~Control hazard~~
    - We do not implement branch instructions now

Data hazard

- You need to detect when the data hazard occurs

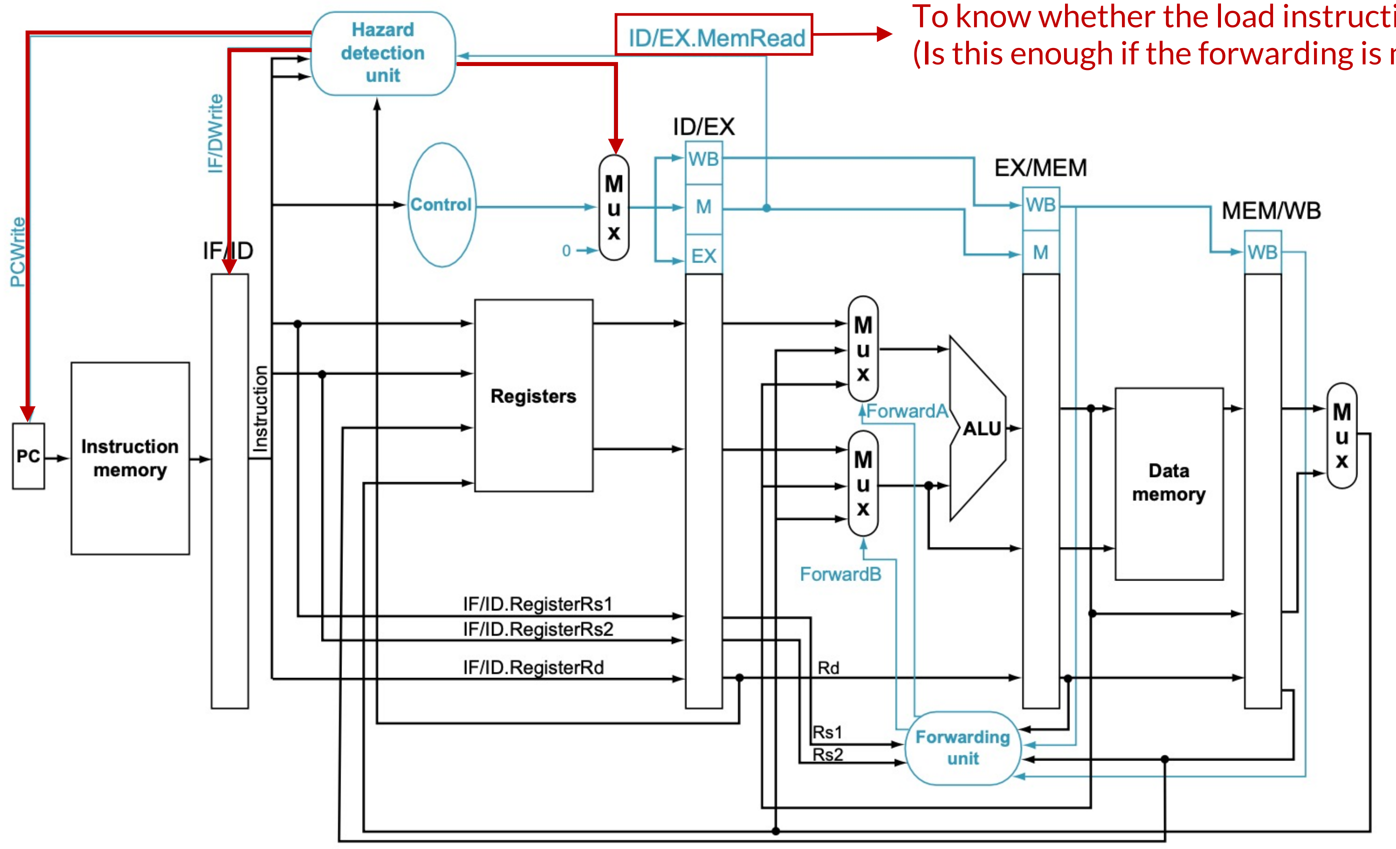


NOP must be inserted
if data forwarding does not occur

NOP must be inserted
if data forwarding does not occur
because your register files is not updated
in the middle of clock cycle

Data hazard

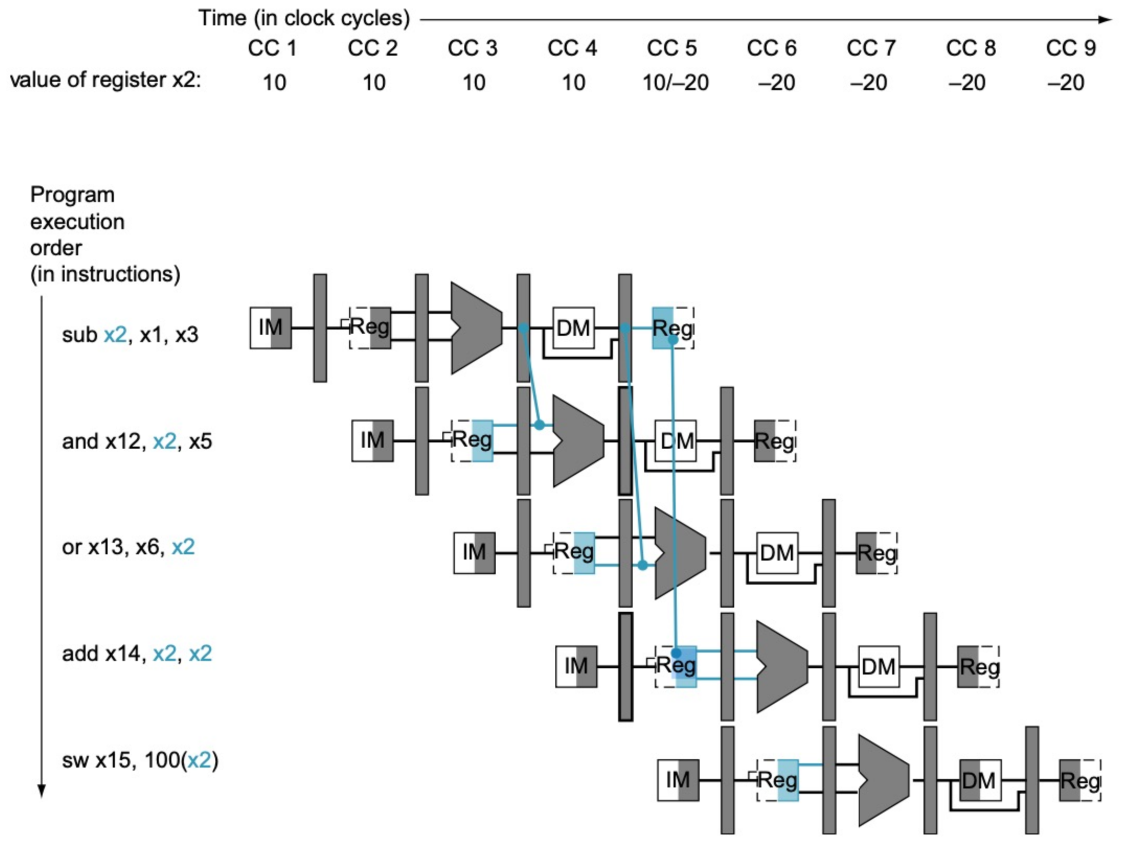
- To stall the pipeline, you need to prevent some registers from being written



To know whether the load instruction is pipelined.
(Is this enough if the forwarding is not implemented?)

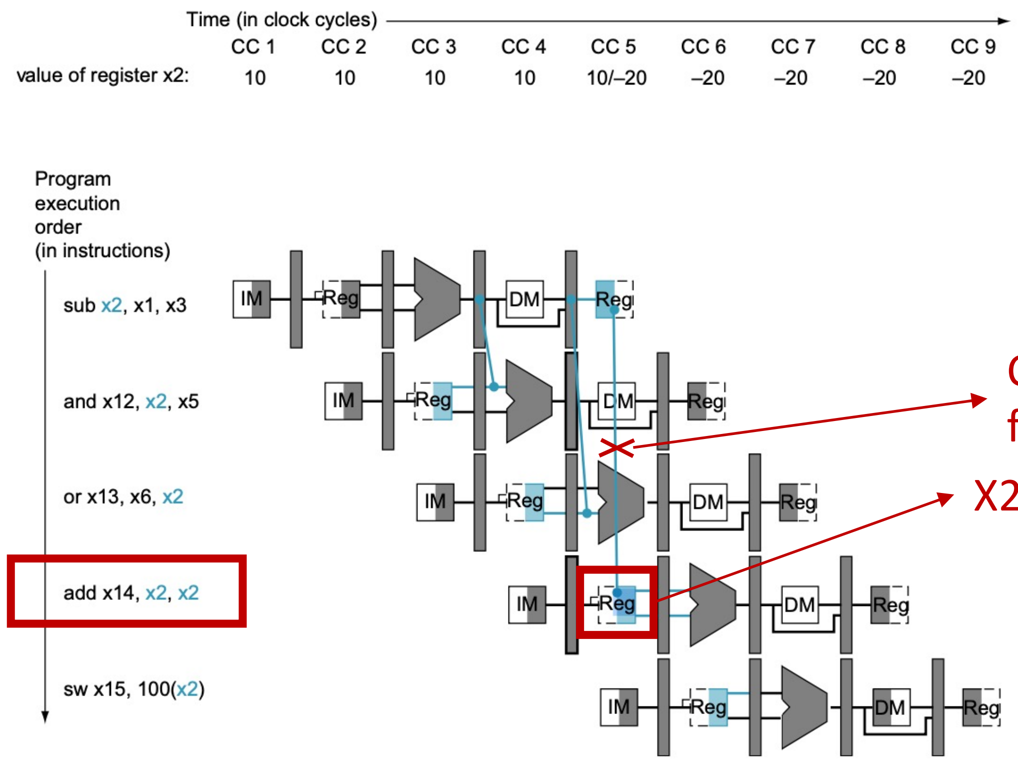
Data hazard

- To reduce stalls, you can also implement data forwarding (extra credit)



Data hazard (forwarding)

- Your register file does not write data into register file in the middle of clock cycle
  - Register x2 read by add x14, x2, x2 is old

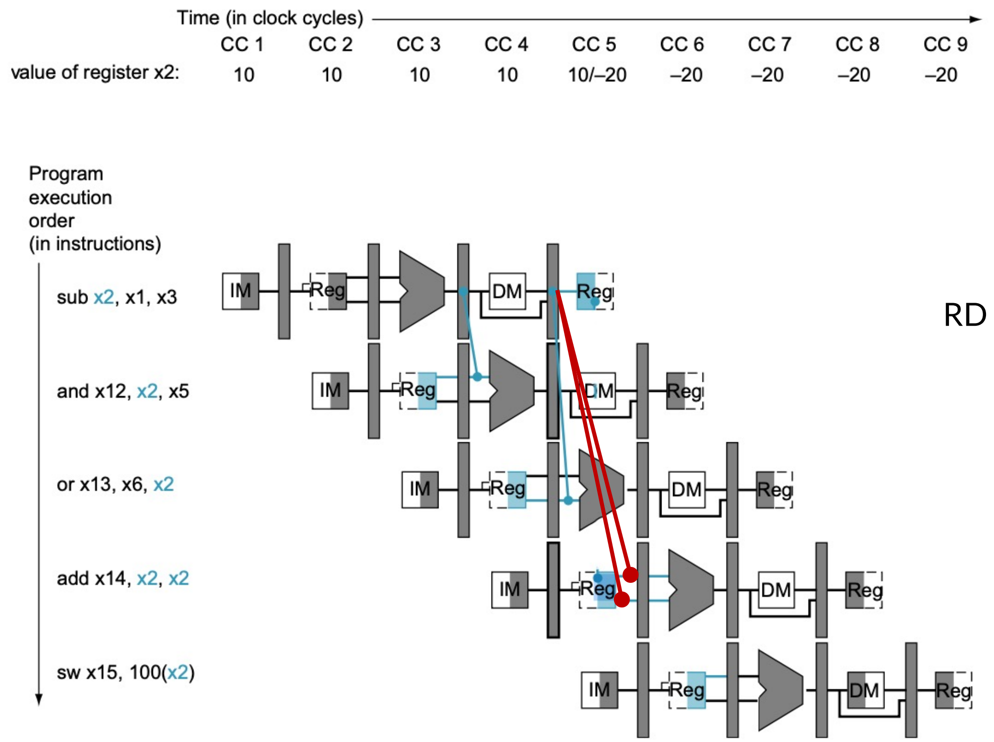

Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
value of register x2: 10 10 10 10 10/–20 –20 –20 –20 –20
Program execution order (in instructions)
sub x2, x1, x3
and x12, x2, x5
or x13, x6, x2
add x14, x2, x2
sw x15, 100(x2)
IM
Reg
DM

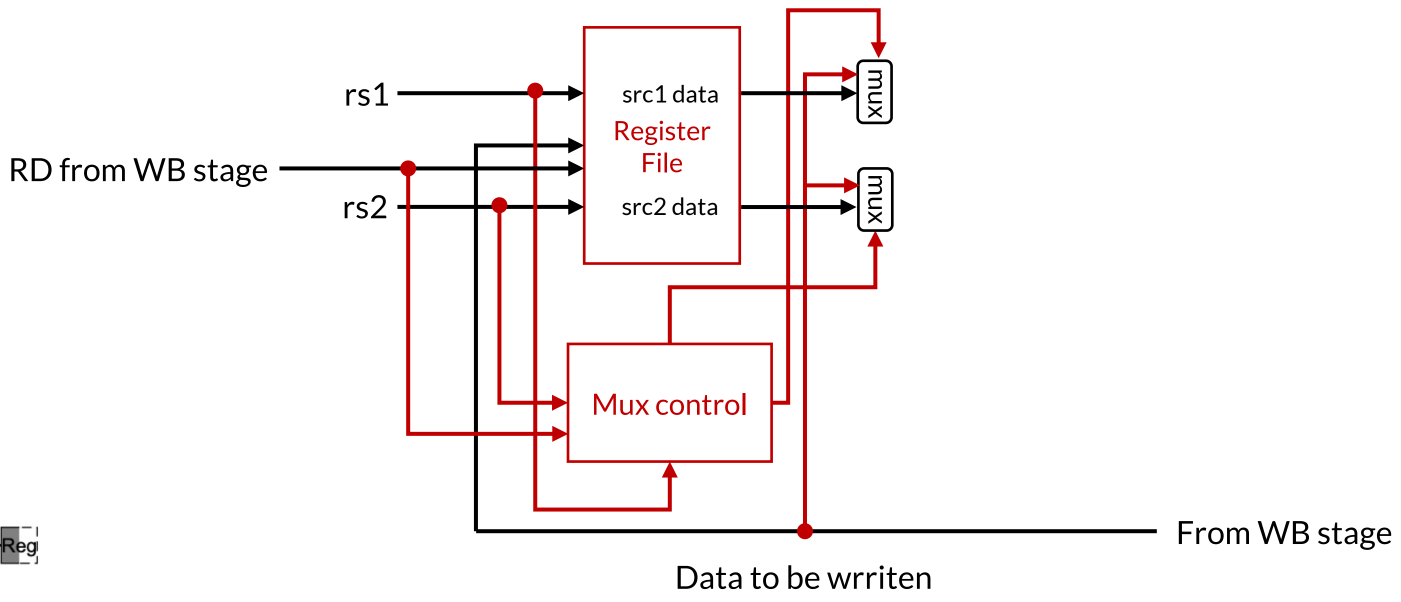
Cannot read newly updated register from the register file

X2 is old!

Data hazard (forwarding)

- Your register file does not write data into register file in the middle of clock cycle
  - We need to forward data from MEM/WB to ID stage

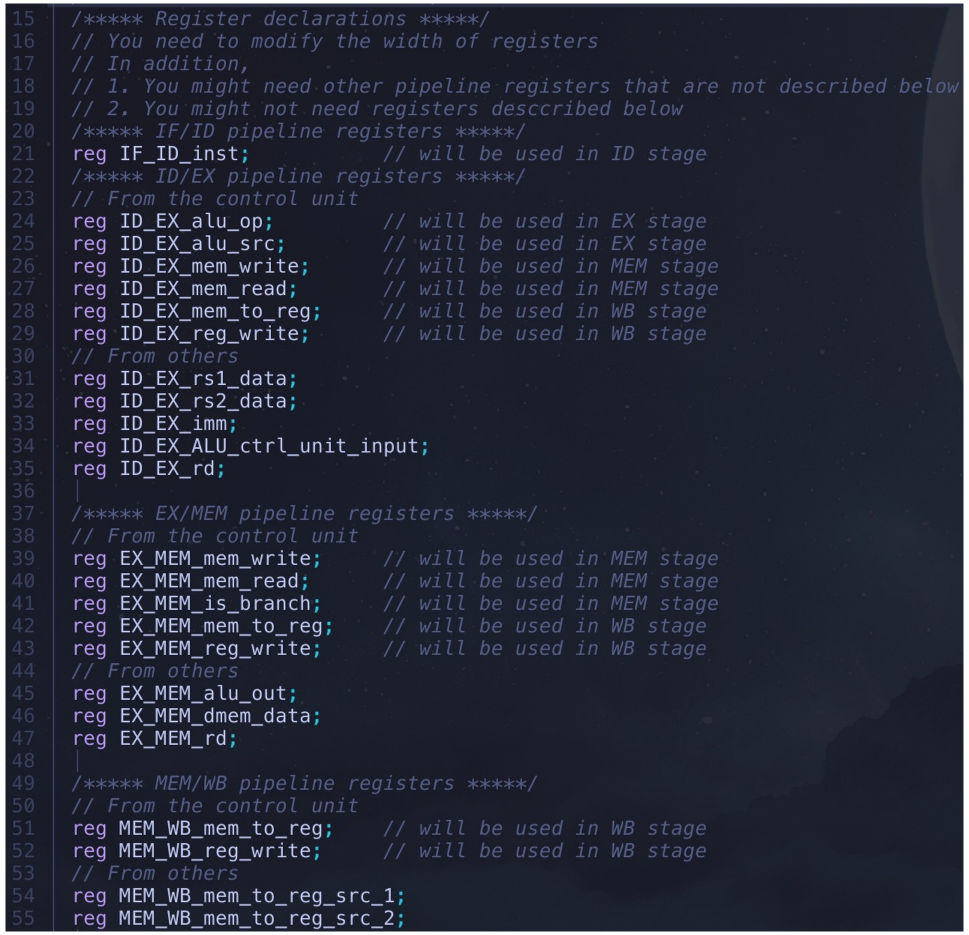




Data hazard (forwarding)

- You might need more pipeline register fields to implement forwarding

You can't implement forwarding with these pipeline registers



```
/***** Register declarations *****/
// You need to modify the width of registers
// In addition,
// 1. You might need other pipeline registers that are not described below
// 2. You might not need registers desccribed below
/***** IF/ID pipeline registers *****/
reg IF_ID_inst;           // will be used in ID stage
/***** ID/EX pipeline registers *****/
// From the control unit
reg ID_EX_alu_op;         // will be used in EX stage
reg ID_EX_alu_src;        // will be used in EX stage
reg ID_EX_mem_write;      // will be used in MEM stage
reg ID_EX_mem_read;       // will be used in MEM stage
reg ID_EX_mem_to_reg;     // will be used in WB stage
reg ID_EX_reg_write;      // will be used in WB stage
// From others
reg ID_EX_rs1_data;
reg ID_EX_rs2_data;
reg ID_EX_imm;
reg ID_EX_ALU_ctrl_unit_input;
reg ID_EX_rd;

/***** EX/MEM pipeline registers *****/
// From the control unit
reg EX_MEM_mem_write;     // will be used in MEM stage
reg EX_MEM_mem_read;      // will be used in MEM stage
reg EX_MEM_is_branch;     // will be used in MEM stage
reg EX_MEM_mem_to_reg;    // will be used in WB stage
reg EX_MEM_reg_write;     // will be used in WB stage
// From others
reg EX_MEM_alu_out;
reg EX_MEM_dmem_data;
reg EX_MEM_rd;

/***** MEM/WB pipeline registers *****/
// From the control unit
reg MEM_WB_mem_to_reg;    // will be used in WB stage
reg MEM_WB_reg_write;     // will be used in WB stage
// From others
reg MEM_WB_mem_to_reg_src_1;
reg MEM_WB_mem_to_reg_src_2;
```

Submission

- Implementation (Deadline: 5/03 9:00 am)
  - 5-stage pipelined CPU
    - Data hazard
      - Stall (no extra credit)
      - Data forwarding (extra credit +5)
    - You don't have to handle control hazard
      - Branch instructions will not be used in this implementation
  - You need to follow the rules described in lab_guide.pdf

- Report (Deadline: 5/03 23:59)
  - How does your pipelined CPU work?
  - Compare total cycles between the single cycle and pipeliend CPU
    - Non-control flow input file
  - How to implement data forwarding?
    - When forwarded?
  - How to implement hazard detection?
    - When detected?

Submission

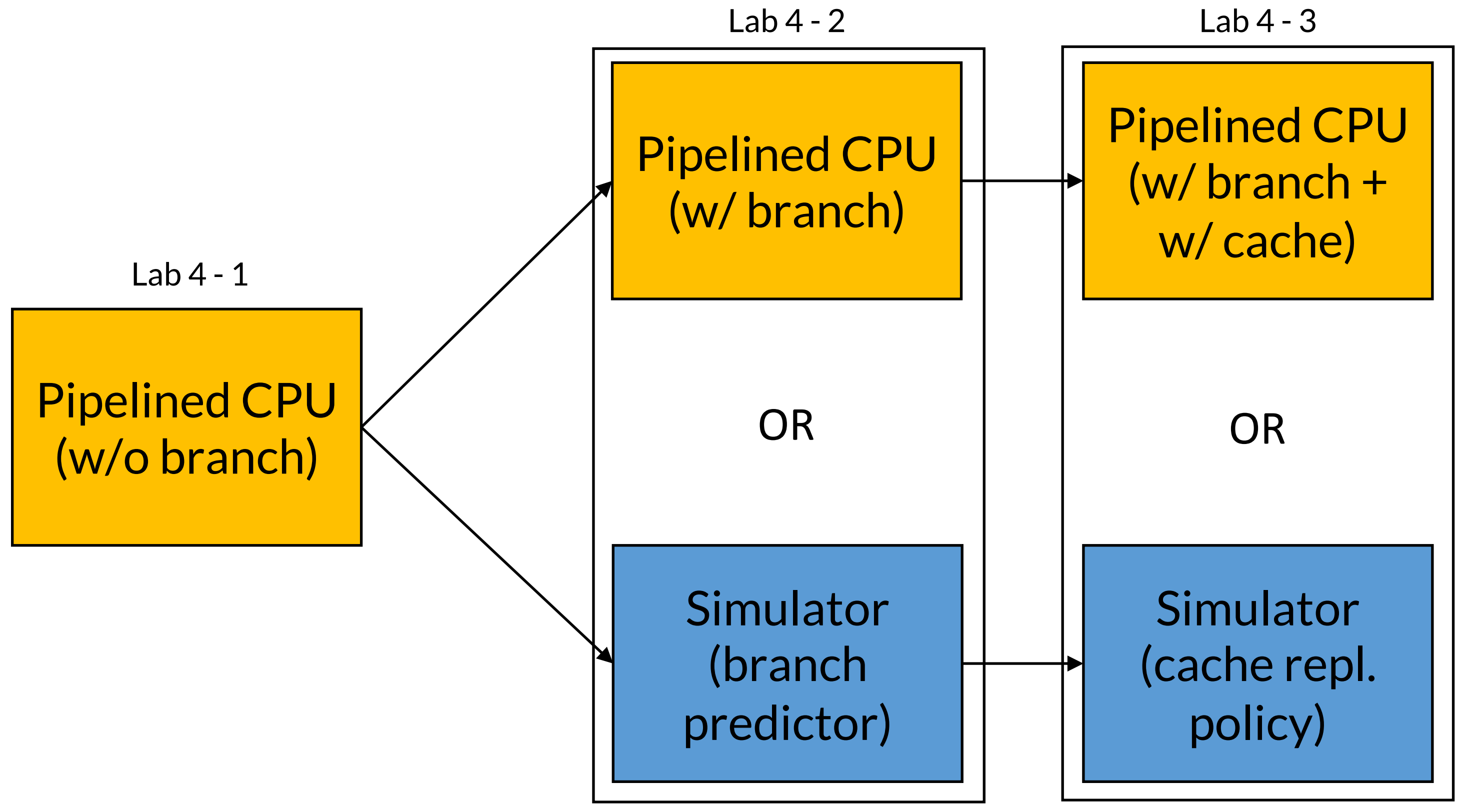
- File format
  - .zip file name: Lab4_{team_num}_{student1_id}_{student2_id}.zip
  - Contents of the zip file (only *.v):
    - cpu.v
    - …
    - Do not include top.v, Memory.v, and RegisterFile.v

Evaluation environment

- Vivado (XSim) will be used to evaluate your submission

Lab schedule

- All students must implement pipelined CPU without branch
  - Students will implement verilog- or simulator-based CPU from Lab 4-2


Lab 4 - 1
Pipelined CPU (w/o branch)
Lab 4 - 2
Pipelined CPU (w/ branch)
OR
Simulator (branch predictor)
Lab 4 - 3
Pipelined CPU (w/ branch + w/ cache)
OR
Simulator (cache repl. policy)

🟨 verilog 🟦 Simulator written in C++