CSED311 Lab4-1: Pipelined CPU w/o control flow instructions

Sangyun Oh

sy.oh@postech.ac.kr

Contact the TAs at cs311-2025ta@postech.ac.kr

Contents

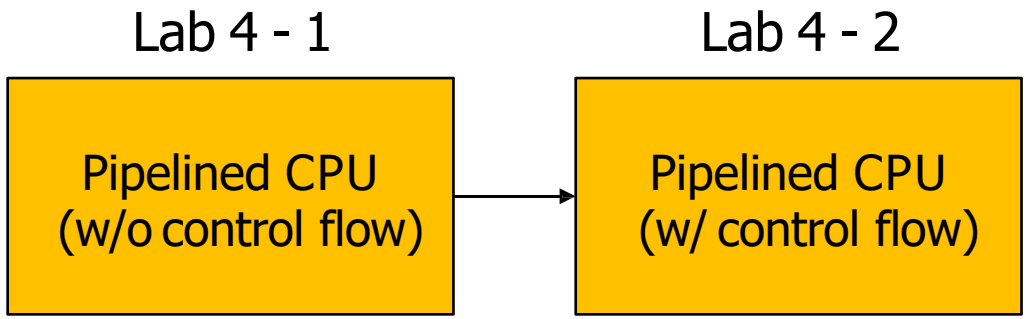
- Objectives
- Pipelined CPU without control flow instructions
- Data Hazard
- Assignment

Objectives

- Understand and implement a pipelined CPU
- First implement a pipelined CPU without control flows

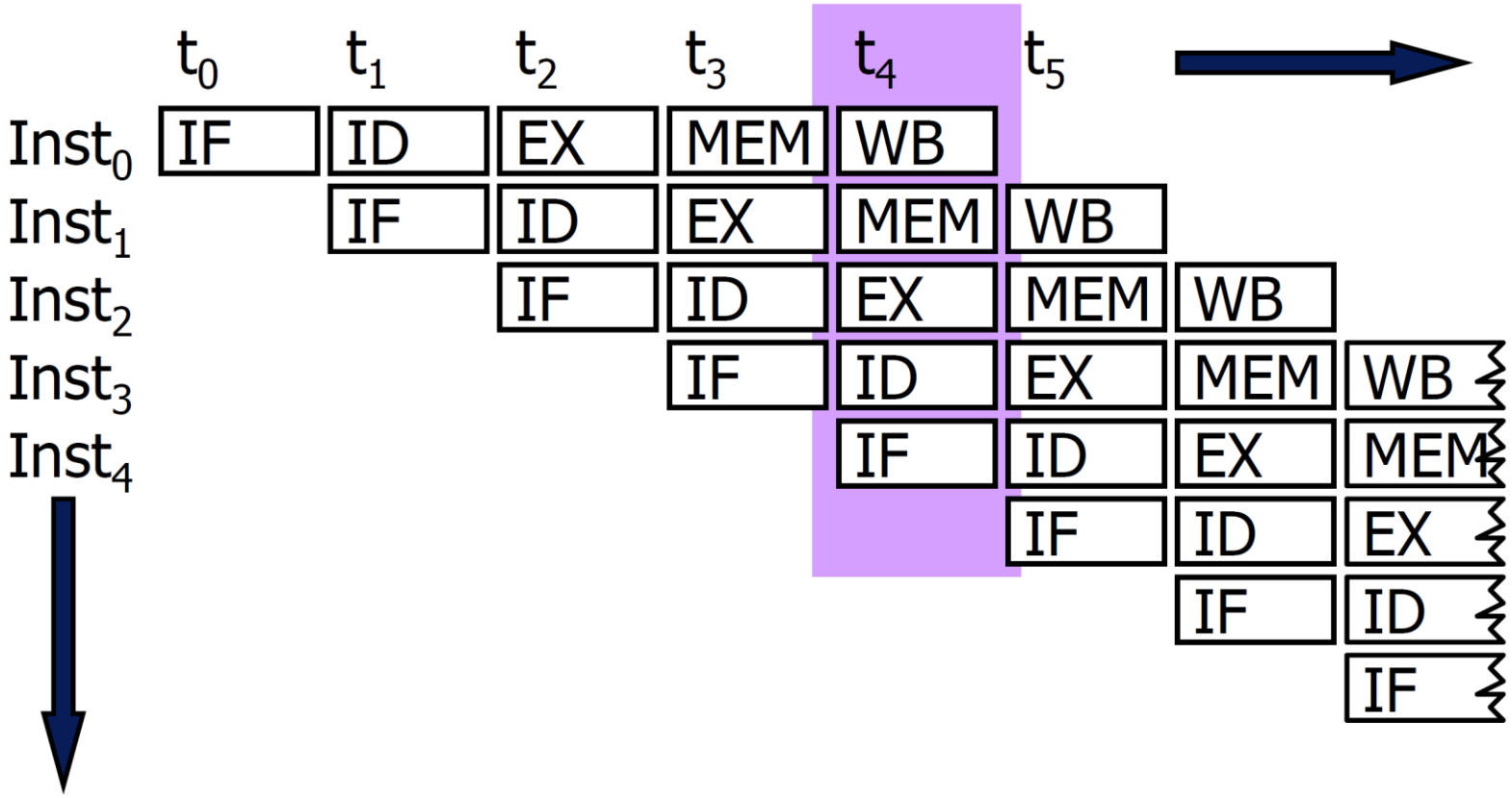
Lab schedule

- You will be implementing a pipelined CPU without control flow instruction support
  - Control flow instructions will be implemented in Lab 4-2



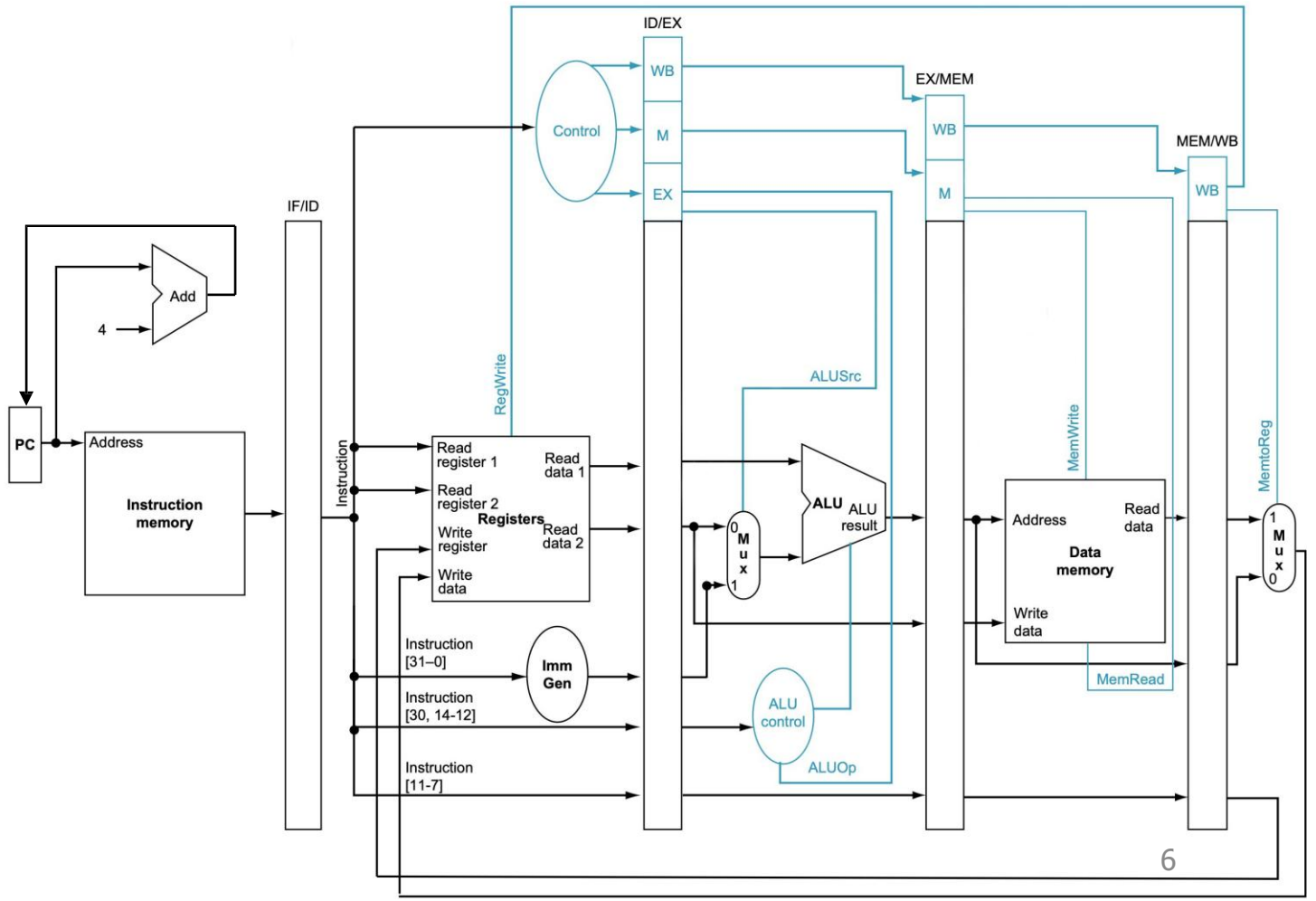
Pipelined CPU

- Increase throughput through better HW utilization



Datapath (w/o control flow instrs.)

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



Update pipeline registers

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

```
/***** 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;
```

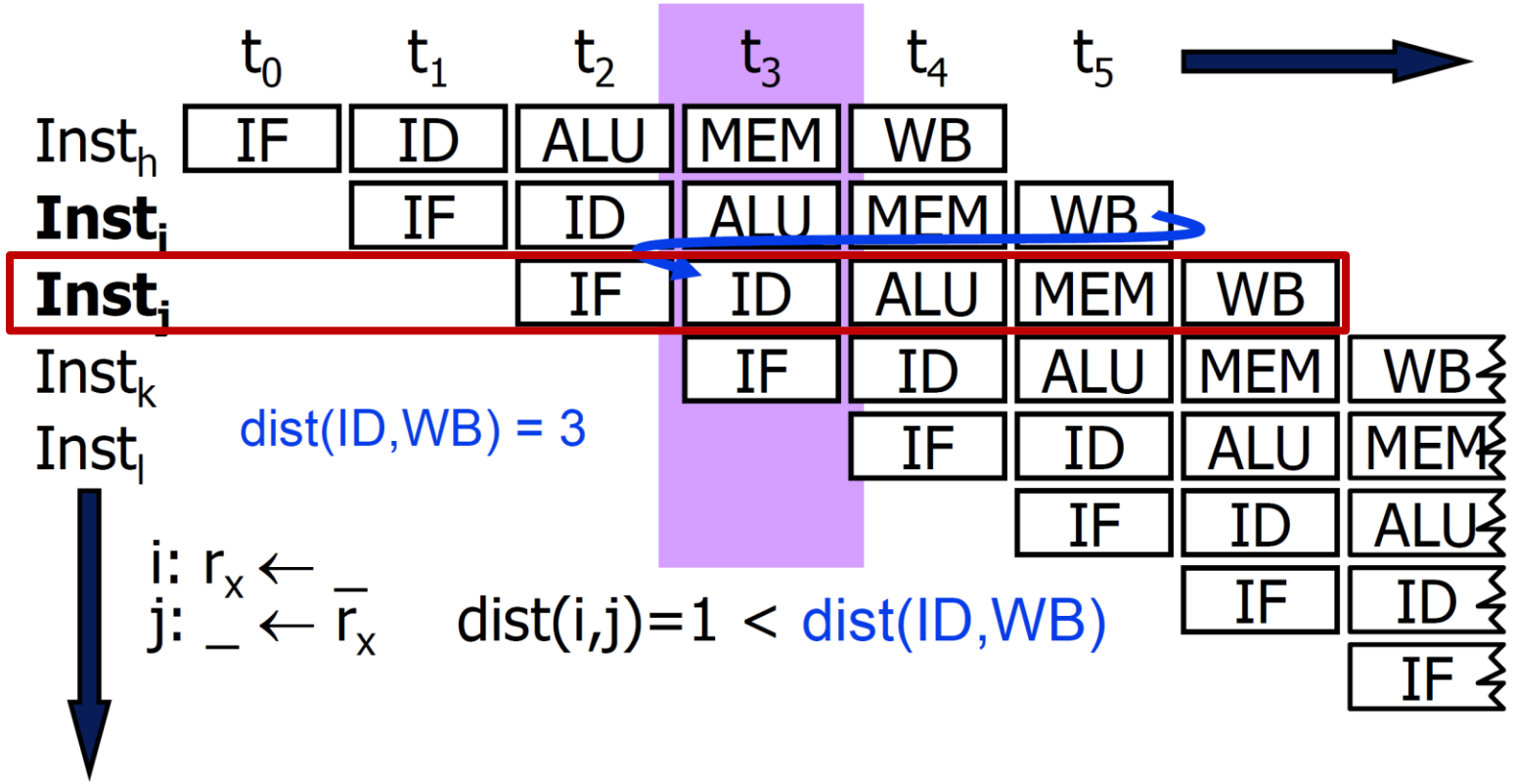
```
// Update ID/EX pipeline registers here
always @(posedge clk) begin
  if (reset) begin
  end
  else begin
  end
end
```

Hazard

- Your implementation should properly 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

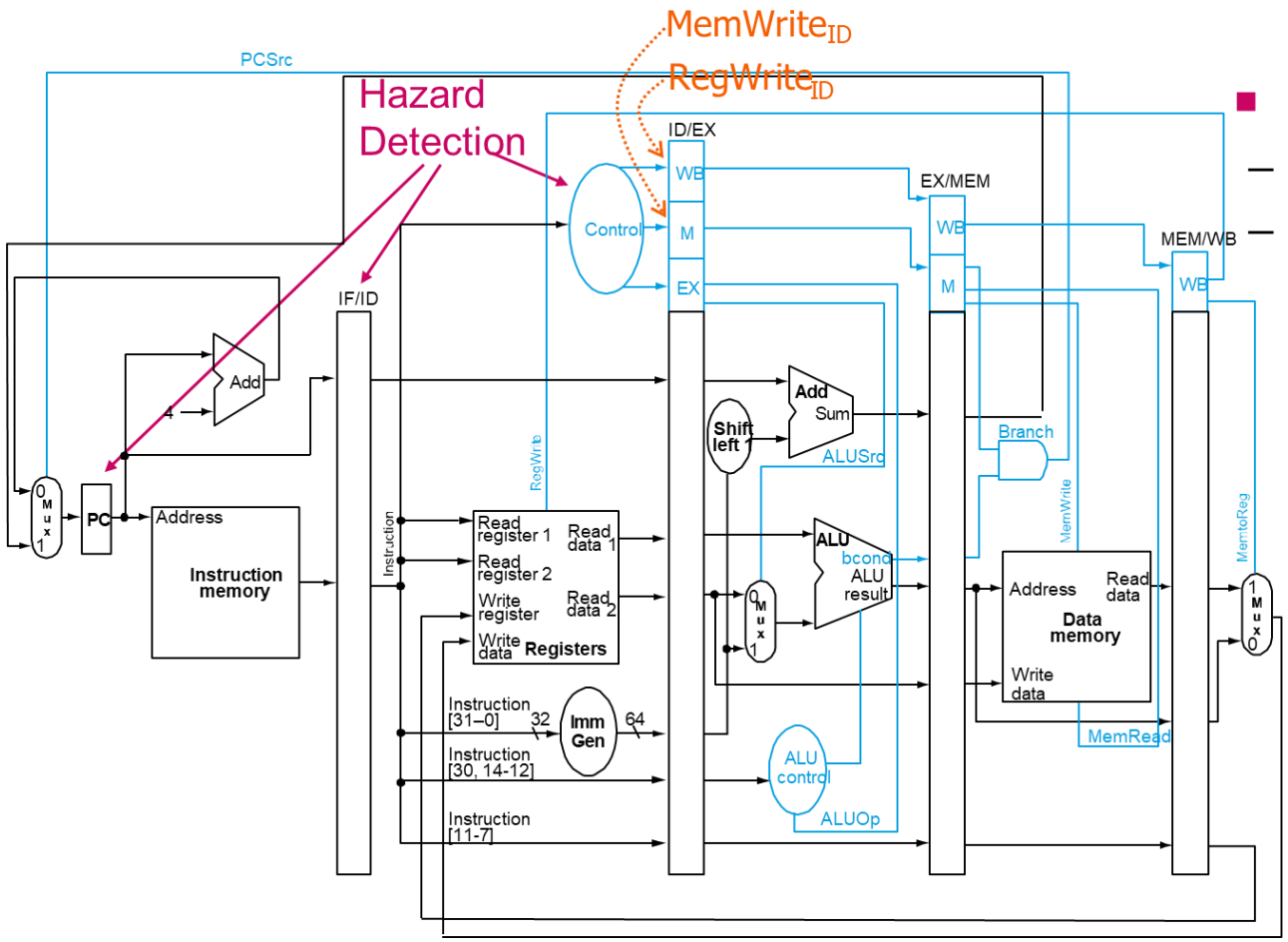
- You need to detect when the data hazard occurs



Inst j should wait until r_x is ready

Data hazard

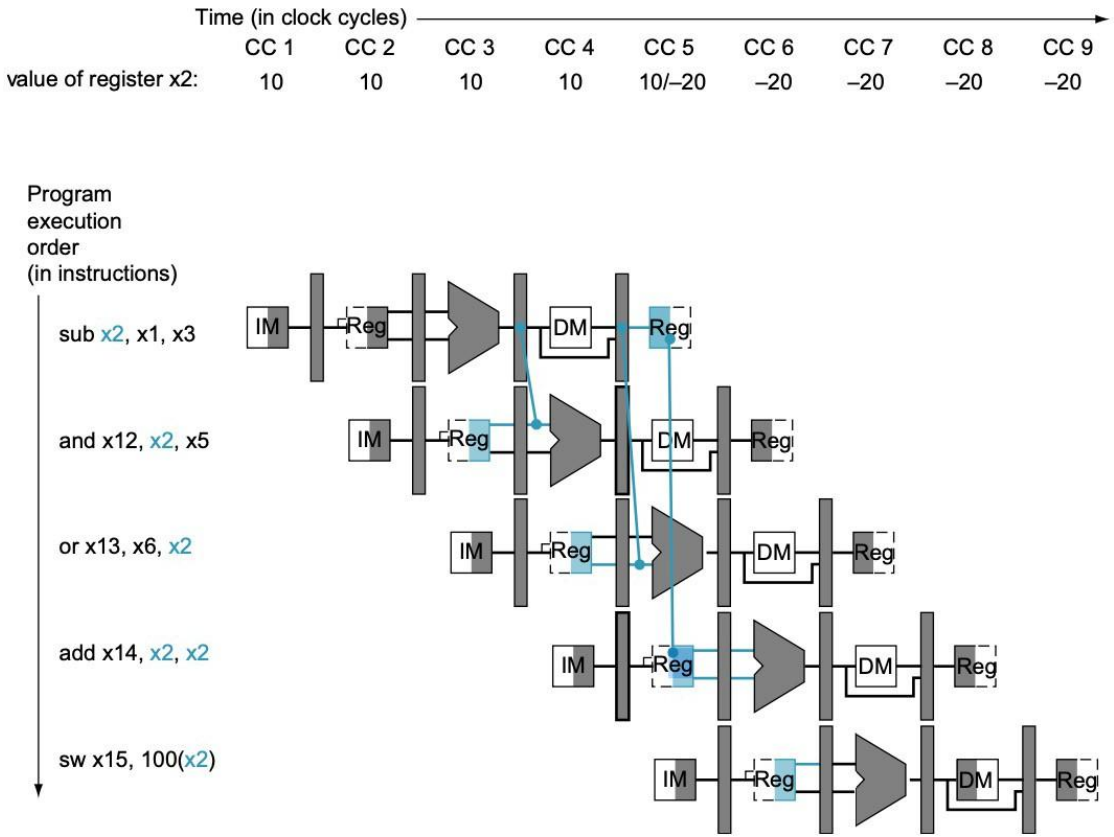
- To stall the pipeline, you need to prevent any architectural state update by NOP(s)



- **Stall**
  - Disable **PC** and **IR** latching
  - Control should set $RegWrite_{ID}$=0 and $MemWrite_{ID}$=0

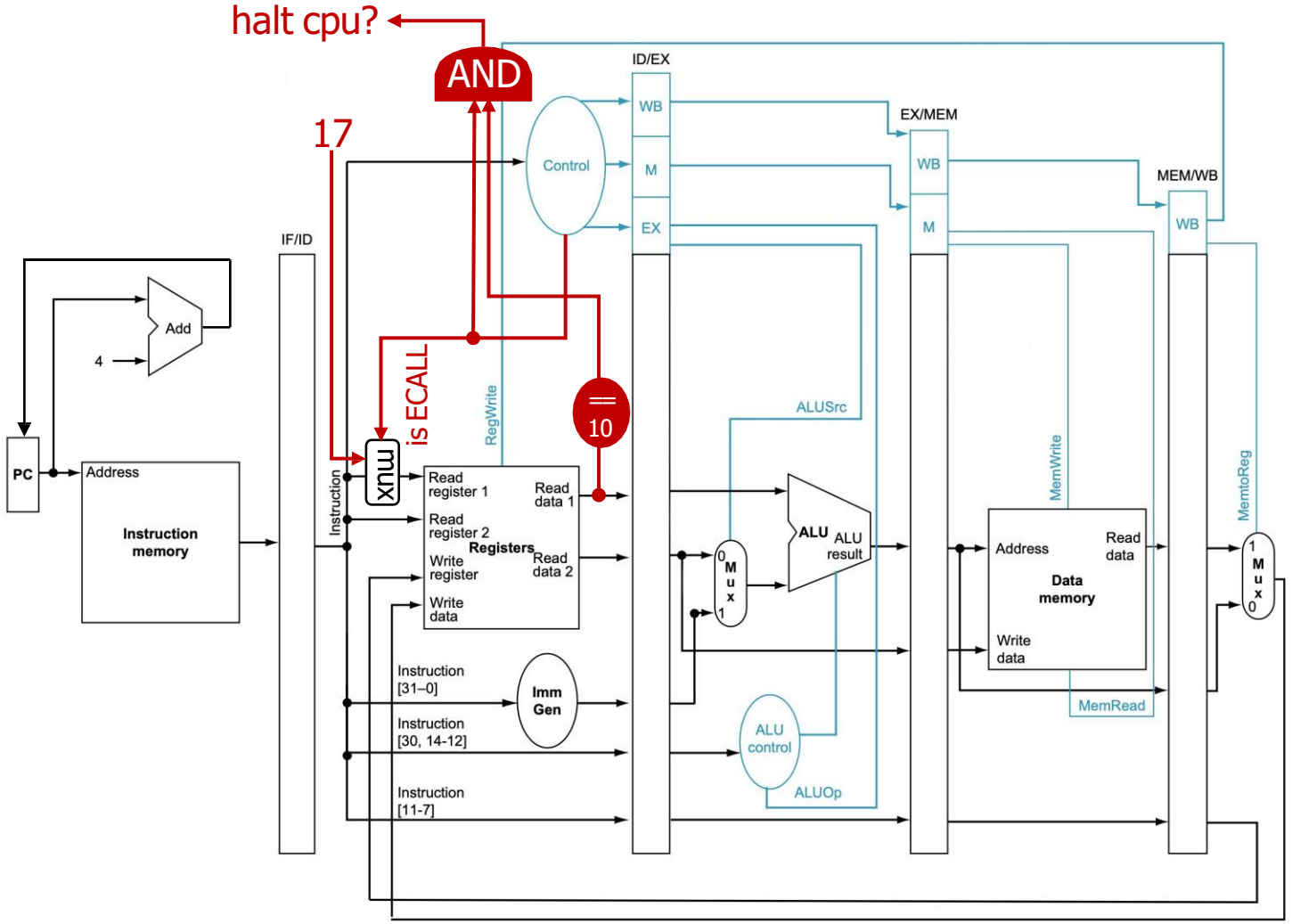
Data forwarding

- To reduce stalls, you can also implement data forwarding
  (necessary for 3-person team w/o extra credit)
  (optional for 2-person team w/ extra credit +10)



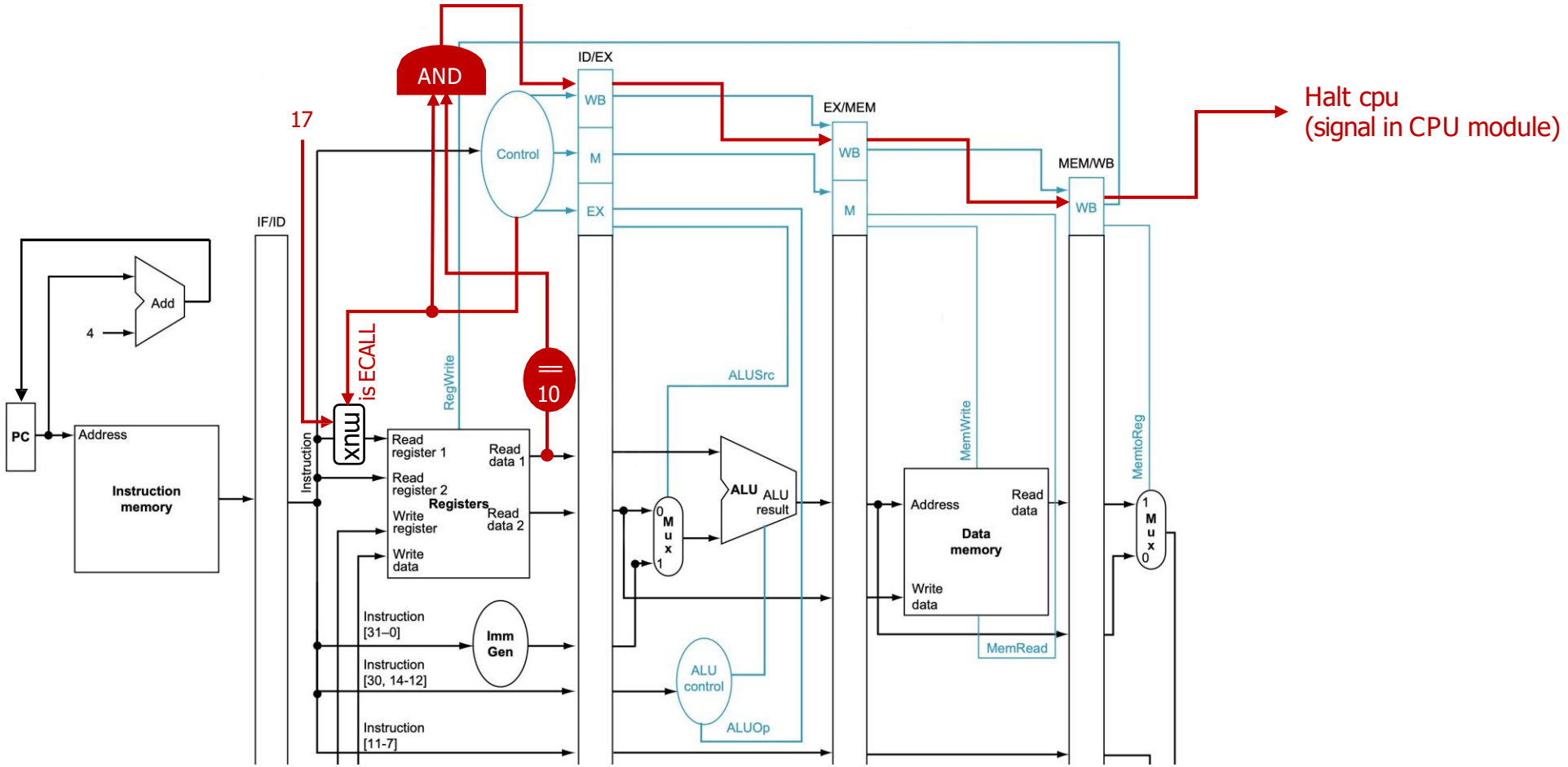
Handling ECALL instruction

- How to halt CPU?



Handling ECALL instruction

- You need to pipeline the halt signal

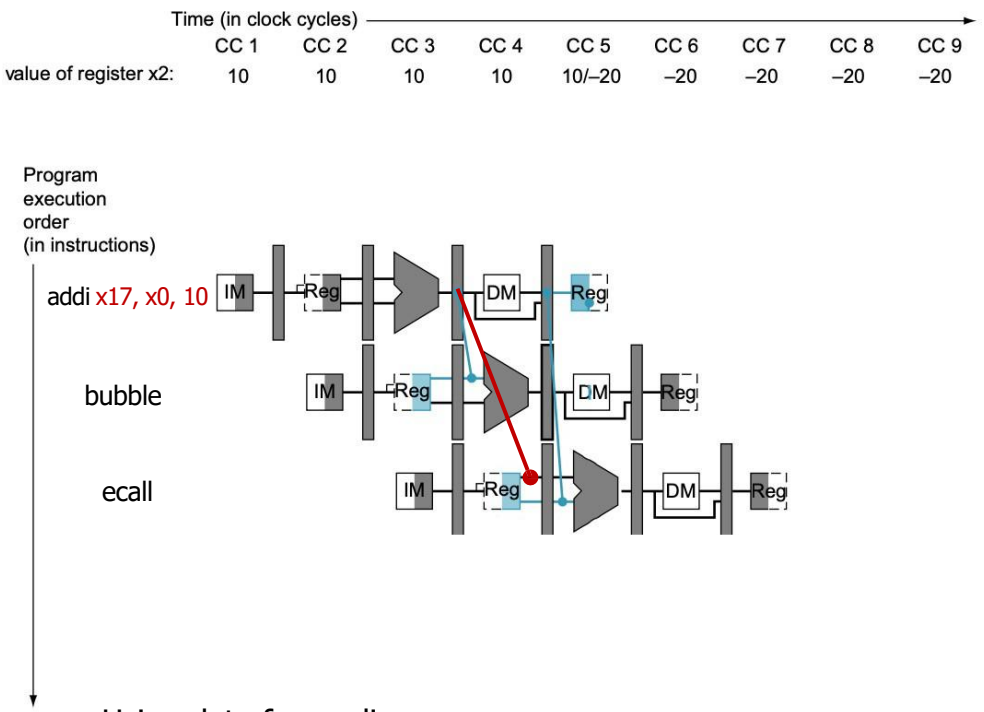


Handling ECALL instruction

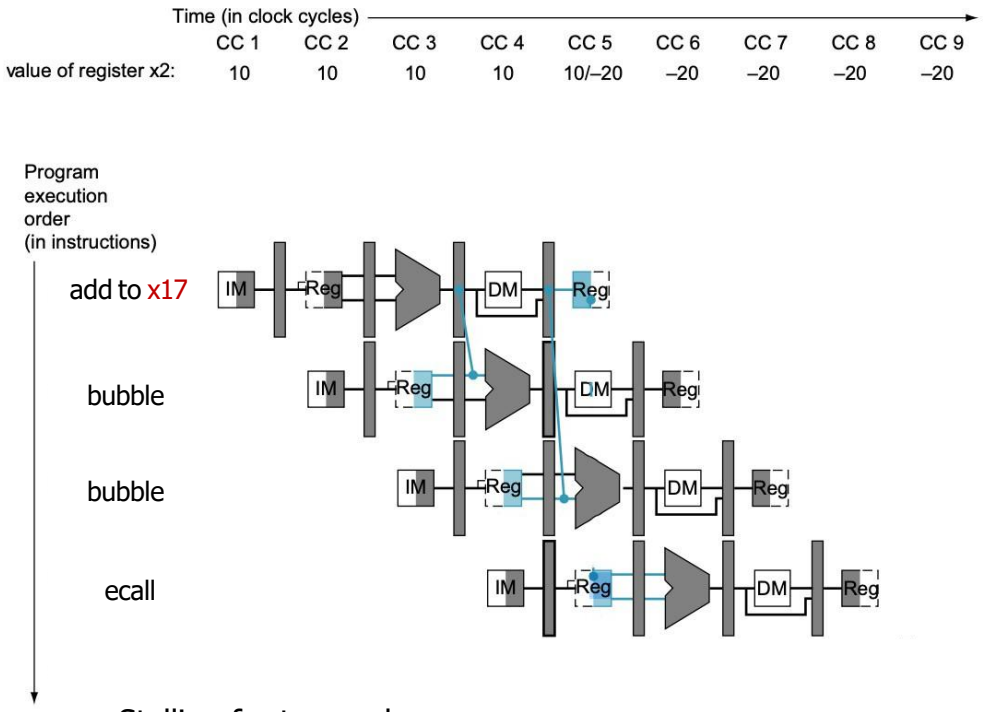
- Hazard and forwarding
  - Also, data hazard should be handled properly by stalls or data forwarding

<executed code>

addi x17, x0, 10
ecall



Using data forwarding



Stalling for two cycles

Submission

- Implementation (Deadline: 4/29 9:00 am)
  - 5-stage pipelined CPU
    - Data hazard
      - Stall (no extra credit)
      - Data forwarding

        necessary for 3-person team

        optional for 2-person team (extra credit +10)
    - You don't have to handle control hazard
      - **Control flow instructions will not be used in this implementation**
  - **You need to follow the rules described in lab_guideline.pdf**
  - Do not modify these files: top.v, RegisterFile.v, DataMemory.v, and InstMemory.v

  - If you wish to present a **demo in the Lab 4-1b session**, please submit your code by April 22nd 9 a.m.
    - (Optional submission) **Code: 2025. 4. 22. 09:00 a.m.**
    - Code resubmitted after the demo will not be accepted.

Submission

- Report (Deadline: 4/29 18:00)

- The report should include **(1) introduction, (2) design, (3) implementation, (4) discussion, and (5) conclusion sections.**

- Key points:
  - How does your pipelined CPU work?
  - Compare total cycles between the single cycle and pipelined CPU
    - Non-control flow input file
  - How to implement hazard detection?
    - When detected?
  - How to implement data forwarding?
    - When forwarded?

Submission

- File format
  - pdf file name: Lab4_{team_id}_{student1_id}_{student2_id}.pdf
    - ascending order of student IDs
    - PDF file of your report
  - code file name: Lab4_{team_id}_{student1_id}_{student2_id}.zip
    - **ascending order** of student IDs
    - Contents of the zip file (only *.v):
      - cpu.v
      - …
      - **Do not** include top.v, DataMemory.v, InstMemory.v, and RegisterFile.v
      - **Do not** create a folder within the zip file

        One directory including all codes when unzipped

ETC.

- In this lab, even if the same register is read from and written to at the same cycle, the internal forwarding mentioned in the textbook is not needed, because the write is done first at the at the negative edge.

Questions?