Enhancement For Lab 4

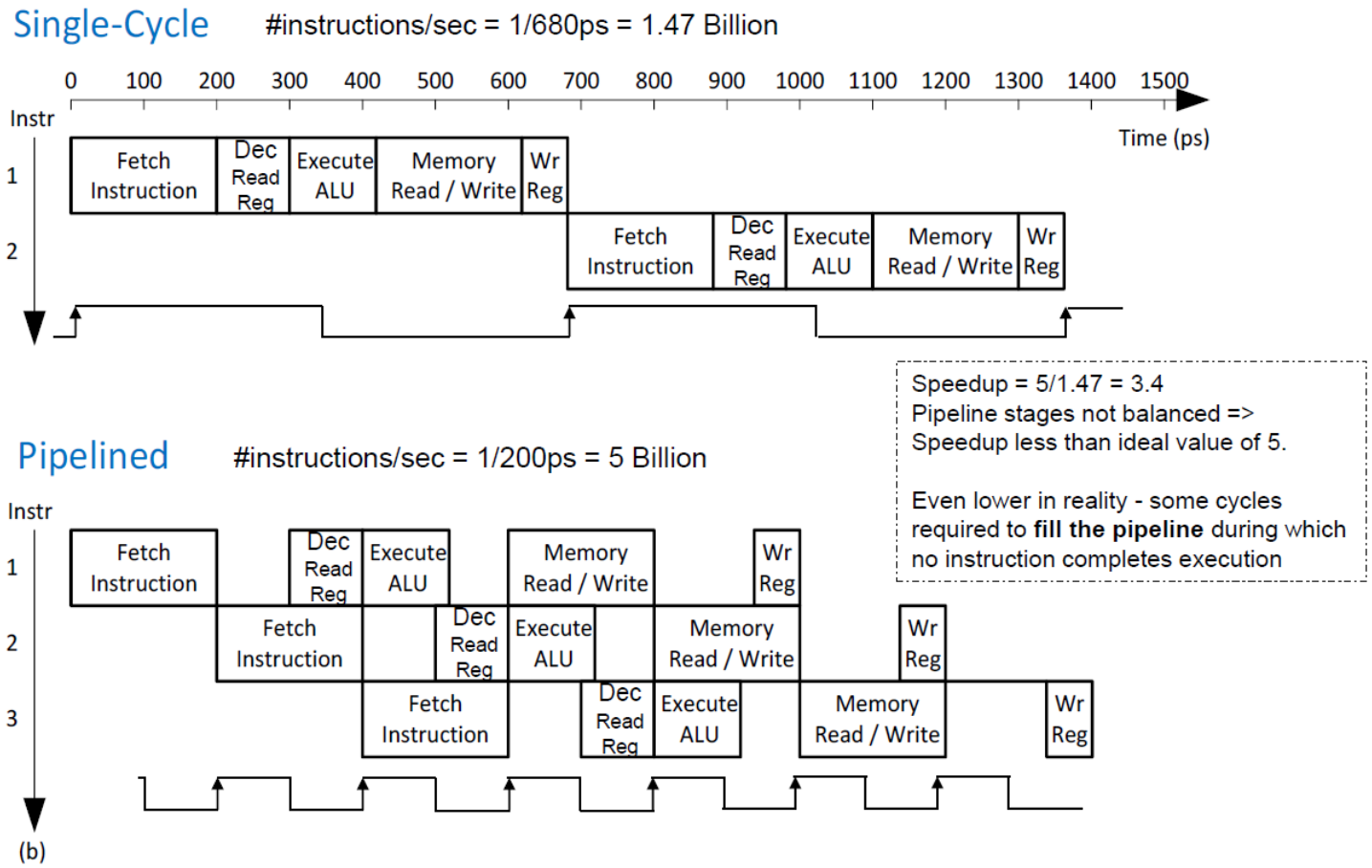
5-Stage Pipelining with Hazard Control Unit

Reason for a Pipelined Processor

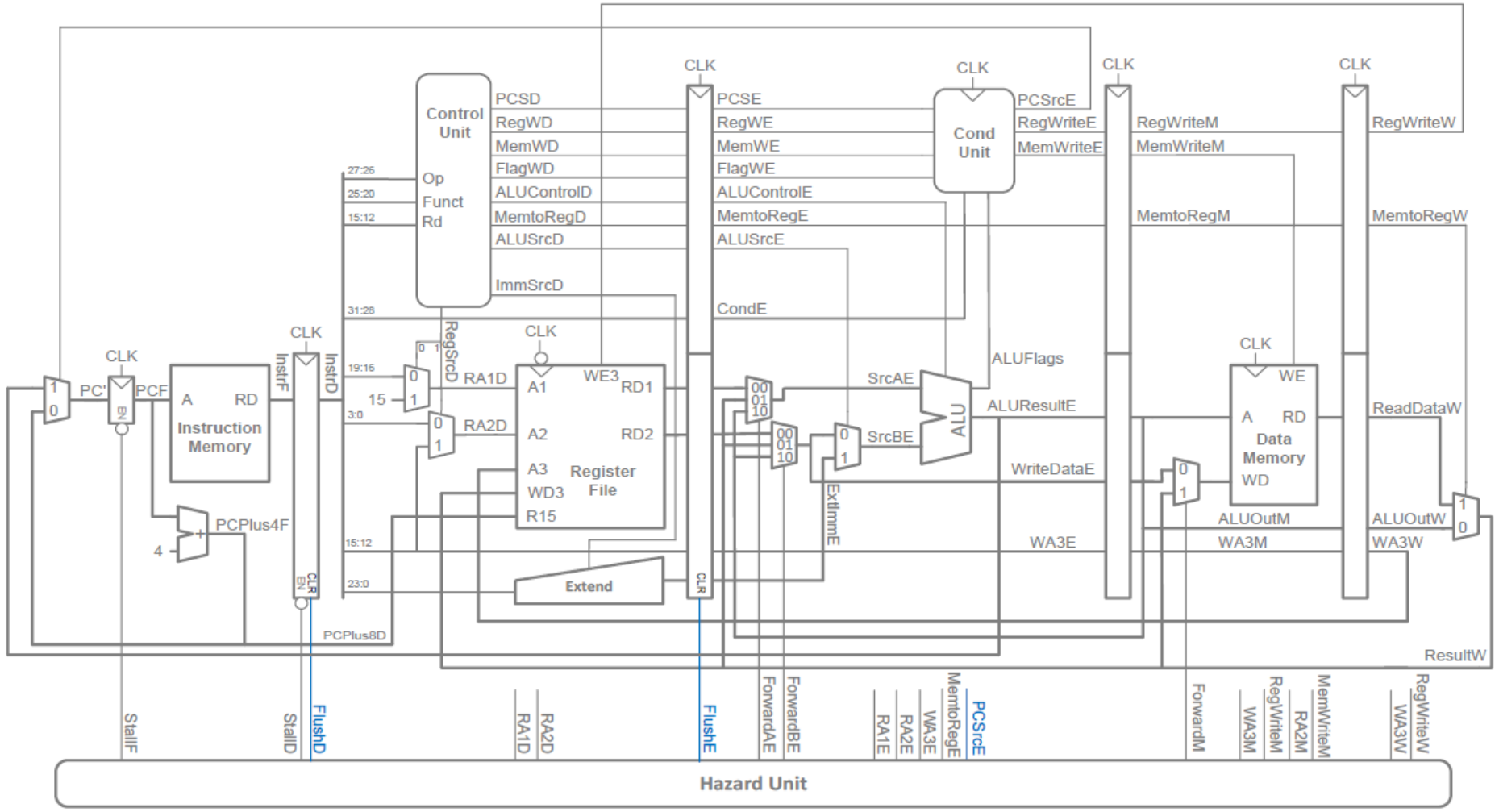
- Pipelining increases performance of the processor, compared to a Single-Cycle Processor.
- A Single-Cycle Processor is slow, as the clock cycle must accommodate the slowest instruction.
  - A LDR instruction takes about 680ps, while a DP Instruction only takes about 480ps
  - The processor has to have a clock cycle of at least 680ps to accommodate for the slowest instruction (200ps wasted on a DP Instruction – Inefficient)

5-Stage Pipelined Processor

- A pipelined processor will improve throughput, reduced clock cycle.
- Starts the next instruction before the current one has completed.



Pipelined Processor with Hazard Control Unit



Hazards due to Pipelining

- Structural Hazards – attempt to use the same resource by two different instructions at the same time

- Data Hazards: Attempt to use data before it is ready (e.g Using data before it has been loaded into a register)

- Control Hazards: Attempt to make a decision about program control flow before the condition has been evaluated and the new PC address calculated (e.g Branch Instructions)

Solution to Structural Hazard

- By using two separate memory in our processor, Instruction Memory and Data Constant Memory, there will be no Structural Hazard in the Processor

Solutions to Data Hazard

- Using Different Clock Edge
  - By using a neg edge CLK for reading registers in RegFile.v, and pos edge for writing data into registers, we can reduce the stalling required by one cycle.
- Data Forwarding
  - Check whether the source register for the instruction in Execute stage matches the destination register in Memory or WriteBack stage
    - Match_1E_M = (RA1E == WA3M), Match_2E_M = (RA2E == WA3M)
    - Match_1E_W = (RA1E == WA3W), Match_2E_W = (RA2E == WA3W)
  - Check if there is a match, and use Data Forwarding through multiplexer to select the right register.

Solutions to Data Hazard

- Memory-memory copy
  - Check if the register used in Memory stage by the STR instruction matches the register written by LDR in the Writeback stage
    - If it is, then forward the result by ForwardM = (RA2M == WA3W) & MemWriteM & MemToRegW & RegWriteW
- Load and Use Hazard
  - There is a problem when a LDR instruction is immediately followed by another instruction that uses the destination register from the LDR instruction
  - The data is only available in the Writeback stage, hence the next instruction that requires the LDR destination register has to be stalled for one cycle.
    - Check Match_12D_E = (RA1D == WA3E) || ((RA2D == WA3E) & ~MemWD)
    - ldrstall = Match_12D_E & MemtoRegE & RegWriteE
    - StallF = StallD = FlushE = ldrstall

Solutions to Control Hazards

- Early BTA
  - Move the Decision point as early in the pipeline as possible, reducing the number of stall cycles.
  - As we will know whether Branching is required at the Execute stage, we can use the PCSrcE value immediately, to change the PC value, either from PC+4, or the ALUResultE. This will result in lesser flush cycles needed.
  - If PCSrcE is true, flush the 2 earlier stages, so that they are not executed
    - FlushD = FlushE = PCSrcE
  - However, this method will not work for LDR R15, .. Instruction