# Pipelined Architecture

## CSEN 3104

## Lecture 7

Dr. Debranjan Sarkar

# Types of parallelism

- Classification proposed by Wolfgang Handler (1977)
- Parallelism and pipelining in 3 distinct levels
  - Processor Control Unit (PCU) =>
    - Processor or CPU
  - Arithmetic Logic Unit (ALU) =>
    - Functional Unit or a processing element (PE) in an array processor
    - An element much smaller than a central processor and having much lower features than a processor
    - working under the control of the processor
    - there are many ALUs in a system, working in parallel to increase the speed of the system
  - Bit Level Circuit (BLC) =>
    - Combinational logic circuit needed to perform the bit operations in the ALU

# 3 Types of pipelining

- Instruction pipelines
  - Instruction Fetching (IF)
  - Instruction decoding (ID)
  - Operand loading (OL)
  - Execution (EX)
  - Operand Storing (OS)
- Arithmetic pipelines
- Processor pipelines
  - a cascade of processors each executing a specific module in the application program

# Hazards of Instruction Pipeline

- Situations preventing execution of the next instruction in the instruction stream during its designated clock cycle
- The instruction is said to be stalled and a hazard is said to exist for that instruction
- The instructions later in the pipeline are also stalled
- Earlier instructions can continue
- No new instructions are fetched during the stall
- Pipeline cannot execute instructions at its peak rate

- 3 types of Hazards
  - (1) Structural hazards
  - (2) Data hazards
  - (3) Control hazards

# Structural Hazards

- A structural hazard refers to a situation in which a required resource is not available (or is busy) for executing an instruction.
- Resource conflicts => Structural hazards
  - use of same resource in different stages
- Resource can be
  - Memory (data and instruction)
  - Functional Unit, which is not fully pipelined
- Example: Let the $i^{th}$ instruction be ADD R4,X
- Here Penalty = 1 cycle

| Clock | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| ADD R4,X | IF | ID | OF | EX | WB | | | | | |
| $(i+1)^{th}$ $inst^n$ | | IF | ID | OF | EX | WB | | | | |
| $(i+2)^{th}$ $inst^n$ | | | Stall | IF | ID | OF | EX | WB | | |
| $(i+3)^{th}$ $inst^n$ | | | | | IF | ID | OF | EX | WB | |

# Structural Hazards

- [1] If an execution unit that requires more than one clock cycle (such as multiply) is not fully pipelined or is not replicated, then a sequence of instructions that uses the unit cannot be subsequently (one per clock cycle) issued for execution.
- Replicating and/or pipelining execution units increases the number of instructions that can be issued simultaneously.
- [2] Another type of structural hazard that may occur is due to the design of register files.
- If a register file does not have multiple write (read) ports, multiple writes (reads) to (from) registers cannot be performed simultaneously.
- For example, under certain situations the instruction pipeline might want to perform two register writes in a clock cycle.
- This may not be possible when the register file has only one write port.
- The effect of a structural hazard can be reduced fairly simply by implementing multiple execution units and using register files with multiple input/output ports.

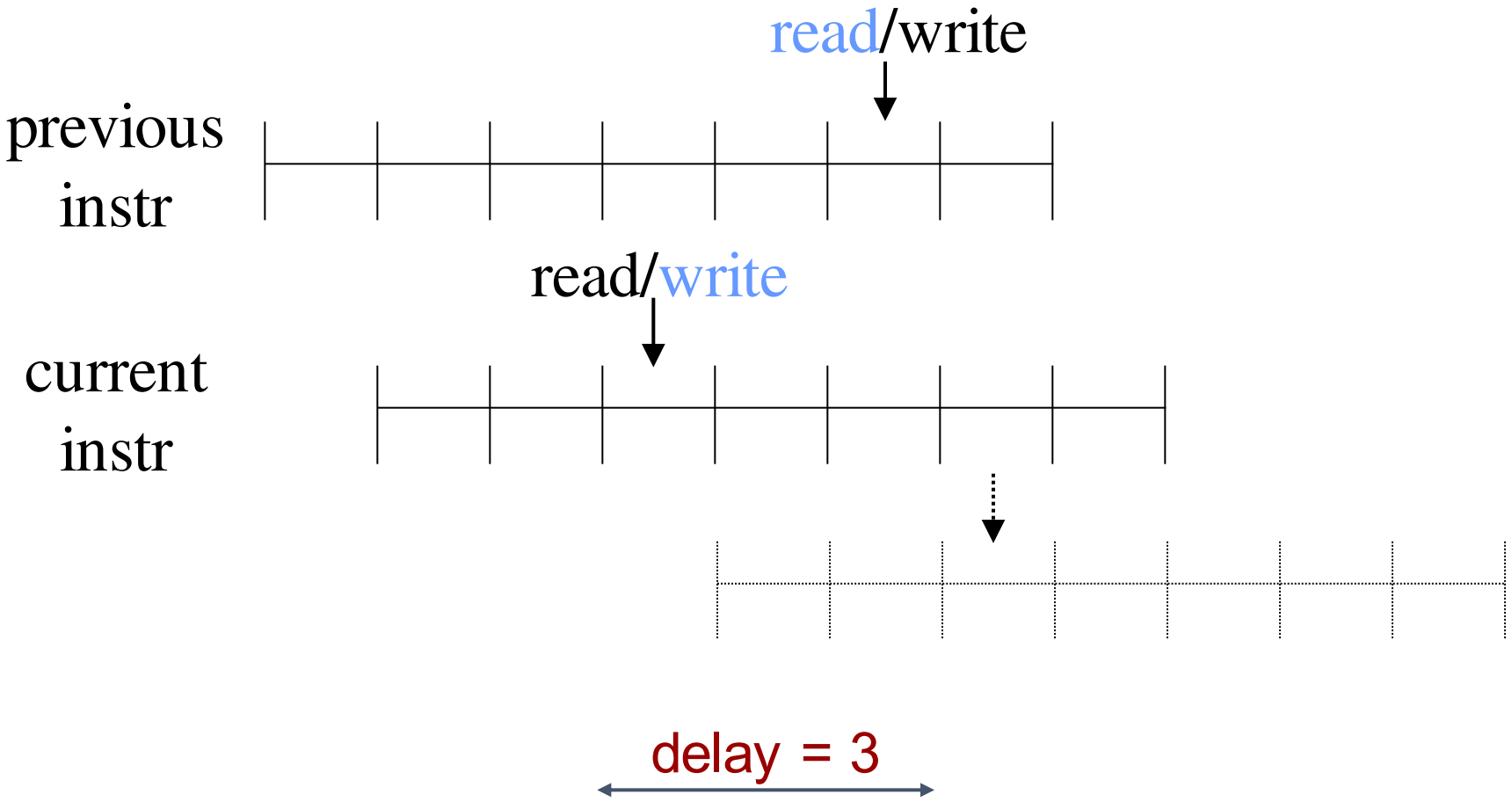
# Techniques to solve Structural Hazards

- Certain resources are duplicated
- Functional Units (ALU, FP unit) can be pipelined themselves in order to support several instructions at a time
- Structural hazard due to memory conflict is avoided by providing separate data cache and instruction cache

# Data Hazards

- A data hazard refers to a situation in which there exists a data dependency (operand conflict) with a previous instruction
- Data dependencies => Data hazards
- Example:
  - Two instructions $I_1$ and $I_2$ are in pipeline
  - The execution of $I_2$ can start before $I_1$ has terminated
  - If $I_2$ needs the result produced by $I_1$, then there is a data hazard
- Three classes of Data hazards
  - RAW (read after write)
  - WAR (write after read)
  - WAW (write after write)
- Read-after-Read (RAR) is not a hazard as no data is changed

# Data Hazards



# Data Hazards: RAW (Read after Write)

- Instruction (I): Write data object X
- Instruction (J): Read data object X
- J tries to read data before it is written by I
- So, J gets old value of data which is incorrect
- Program order must be preserved to ensure that J gets the correct data value

# Data Hazards: WAW (Write after Write)

- Instruction (I): Write data object X
- Instruction (J): Write (or modify) data object X
- J tries to write (modify) data before it is written by I
- So, finally the data object written by instruction I prevails which is not desirable
- It was desired to have the final value of data object X written by instruction J and not by instruction I
- Program order must be preserved to ensure that J gets the correct data value

# Data Hazards: WAR (Write after Read)

- Instruction (I): Read data object X
- Instruction (J): Write (or modify) data object X
- J tries to write (modify) data object before it is read by I
- So, the data object read by instruction I is incorrect
- It was desired that instruction I would read the old value of X and then the data object would be modified by instruction J
- Program order must be preserved to ensure that J gets the correct data value

Thank you