

MODULE 3

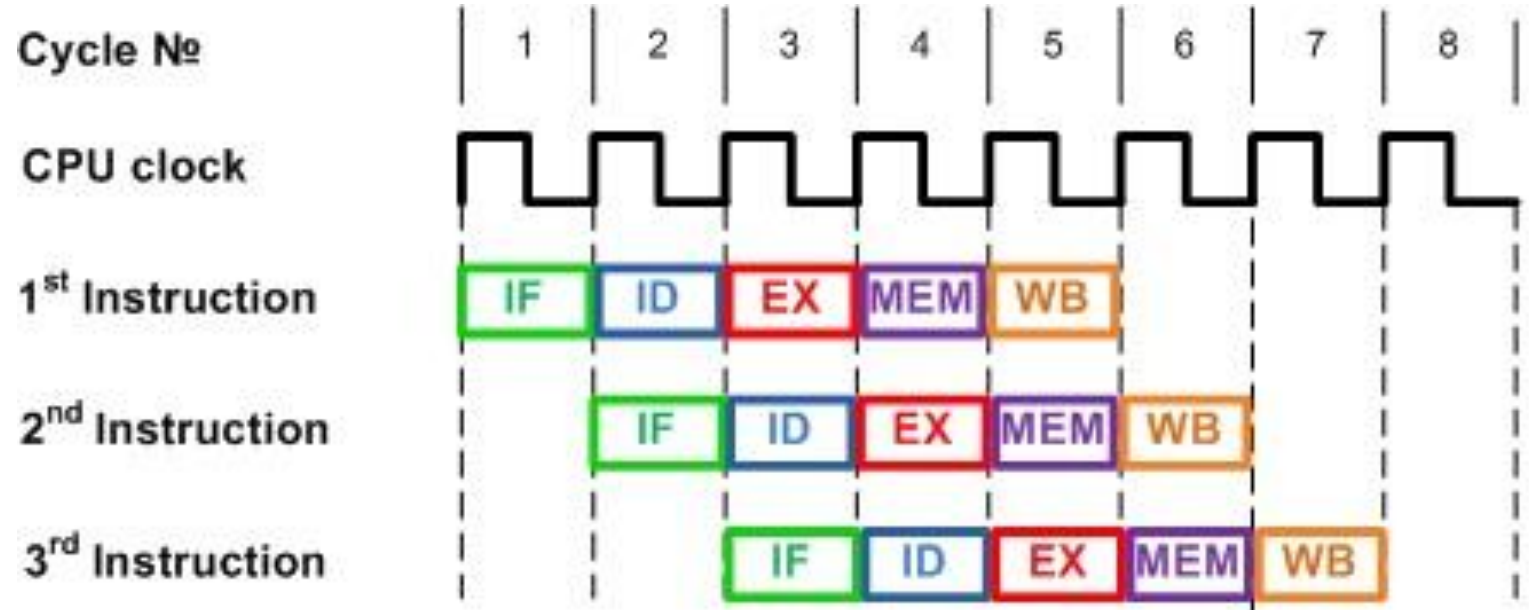
PIPELINING – BASIC PRINCIPLES

**CST 202 : Computer Organization
& Architecture**

Pipelining- Basic Principles

- Pipelining is a method to realize, **overlapped parallelism** in the proposed solution of a problem, on a digital computer in an economical way.
- Pipelining is the method to introduce temporal parallelism in computer operations.
- A pipeline processor may process each instruction in 5 steps:
 - **F Fetch:** Read the instruction from the memory
 - **D Decode:** Decode the instruction and fetch the source operands
 - **E Execute:** Perform the operation specified by the instruction
 - **M Memory Access:** In case if the instruction demands
 - **W Write:** Store the result in the destination location.

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Execute (EX)
- Memory Access (MEM)
- Writeback (WB)



Performance Improvement

- In order to properly identify the performance improvement that can be achieved using the pipelining technique, we need to use two terms – **latency** and **throughput**.
 - ***Latency** is the time it takes for an operation to complete.*
 - ***Throughput** is the number of operations that are completed in a certain time frame.*
- We can conclude with previous diagram
 - The pipeline does not affect the latency. The instruction cycle consisting of 5 phases is the same whether pipelining is used or not. Each instruction takes 5 clock cycles to complete.
 - The pipeline improves the throughput, with potential speedup equal to the number of pipeline stages.
- Once the pipeline is filled and continuously fed, we can have an instruction completed on every clock cycle (**Cycles Per Instruction (CPI) = 1**). This is, of course, the ideal case not taking into account some of the pipeline limitations

Pipeline Limitations

- There are some known limitations to pipelining technique used in microprocessors. Hazards prevent the next instruction in the pipeline to be executed in its designated clock cycle. The pipeline hazards can be grouped into three categories:
 - **Data Hazard** – Can occur when an instruction depends on the result of a previous instruction still being processed in the pipeline.
 - **Structural Hazard** – Can occur when the available hardware does not support some instruction combinations.
 - **Control Hazards** – Caused by jump and branch instructions. These instructions will usually require the flushing (emptying) of the pipeline and loading it with instructions from addresses pointed by the branch and jump instructions.



CLASSIFICATION OF PIPELINE PROCESSORS

Pipelining- Classification

□ Various types of pipelining can be applied in computer operations depending on the following factors:

- Level of Processing
- Pipeline configuration
- Type of Instruction and data

□

Classification according to level of processing

1. Instruction Pipeline:

- An instruction cycle may consist of many operations like, fetch opcode, decode opcode, compute operand addresses, fetch operands, and execute instructions.
- These operations of the instruction execution cycle can be realized through the pipelining concept.
- Each of these operations forms one stage of a pipeline.
- The overlapping of execution of the operations through the pipeline provides a speedup over the normal execution.
- Thus, the pipelines used for instruction cycle operations are known as instruction pipelines.

Classification according to level of processing

2. Arithmetic Pipeline:

- The complex arithmetic operations like multiplication, and floating point operations consume much of the time of the ALU.
- These operations can also be pipelined by segmenting the operations of the ALU and as a consequence, high speed performance may be achieved.
- Thus, the pipelines used for arithmetic operations are known as arithmetic pipelines.

Classification according to pipeline configuration:

- **Uni-function Pipelines:** When a fixed and dedicated function is performed through a pipeline, it is called a Uni-function pipeline.
- **Multifunction Pipelines:** When different functions at different times are performed through the pipeline, this is known as Multifunction pipeline.
 - Multifunction pipelines are reconfigurable at different times according to the operation being performed.

Classification according to type of instruction and data:

- **Scalar Pipelines:** This type of pipeline processes scalar operands of repeated scalar instructions.
- **Vector Pipelines:** This type of pipeline processes vector instructions over vector operands.

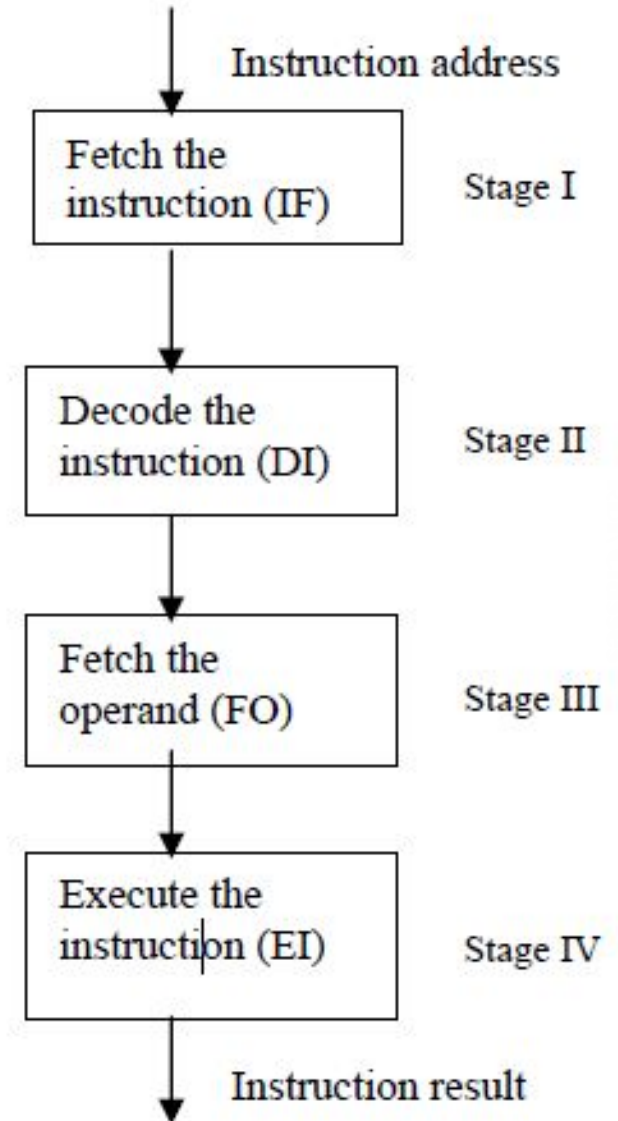


INSTRUCTION AND ARITHMETIC PIPELINES

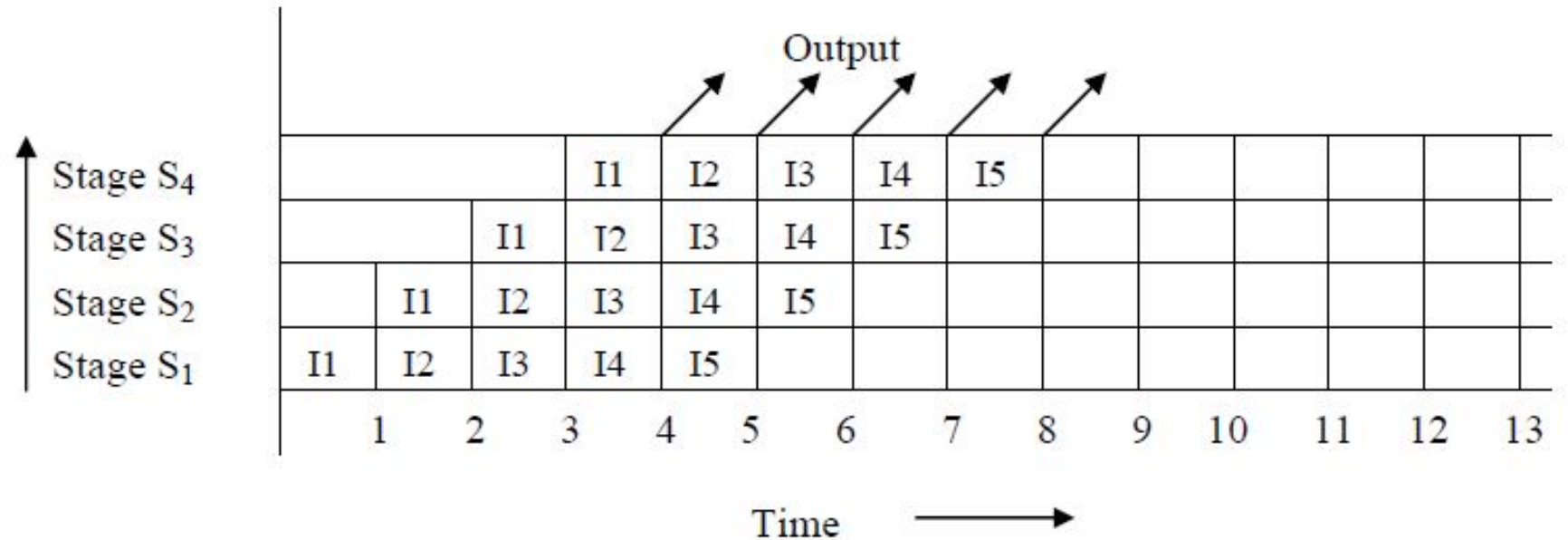
Instruction Pipelining

13

- The stream of instructions in the instruction execution cycle, can be realized through a pipeline where overlapped execution of different operations are performed.
- The process of executing the instruction involves the following major steps:
 - Fetch the instruction from the main memory
 - Decode the instruction
 - Fetch the operand
 - Execute the decoded instruction
- The pipeline is executing the instruction in an overlapped manner increasing the throughput and speed of execution.



14



Instruction Buffers

- For taking the full advantage of pipelining, pipelines should be filled continuously.
- Therefore, **instruction fetch rate** should be matched with the **pipeline consumption rate**. To do this, **instruction buffers** are used.
- Instruction buffers in CPU have **high speed memory** for **storing the instructions**.
- The instructions are pre-fetched in the buffer from the main memory.
- Another alternative for the instruction buffer is the **cache memory** between the CPU and the main memory.
- The advantage of cache memory is that it can be used **for both instruction and data**.
- But cache **requires more complex control logic** than the instruction buffer.

Arithmetic Pipelining

- The technique of pipelining can be applied to various **complex** and **slow** arithmetic operations to **speed up** the **processing time**. And are called **Arithmetic pipelines**.
- Arithmetic pipelines are constructed for **simple fixed-point** and **complex floating-point arithmetic operations**.
 - These arithmetic operations are well suited to pipelining as these operations can be efficiently partitioned into **subtasks** for the **pipeline stages**.
- For implementing the arithmetic pipelines we generally use following two types of adder:
 - **Carry propagation adder (CPA)**: It adds two numbers such that carries generated in successive digits are propagated.
 - **Carry save adder (CSA)**: It adds two numbers such that carries generated are not propagated rather these are saved in a carry vector

Fixed Arithmetic Pipelining- example of multiplication

$$X_5 \ X_4 \ X_3 \ X_2 \ X_1 \ X_0 = X$$

$$Y_5 \ Y_4 \ Y_3 \ Y_2 \ Y_1 \ Y_0 = Y$$

$$X_5Y_0 \ X_4Y_0 \ X_3Y_0 \ X_2Y_0 \ X_1Y_0 \ X_0Y_0 = P_1$$

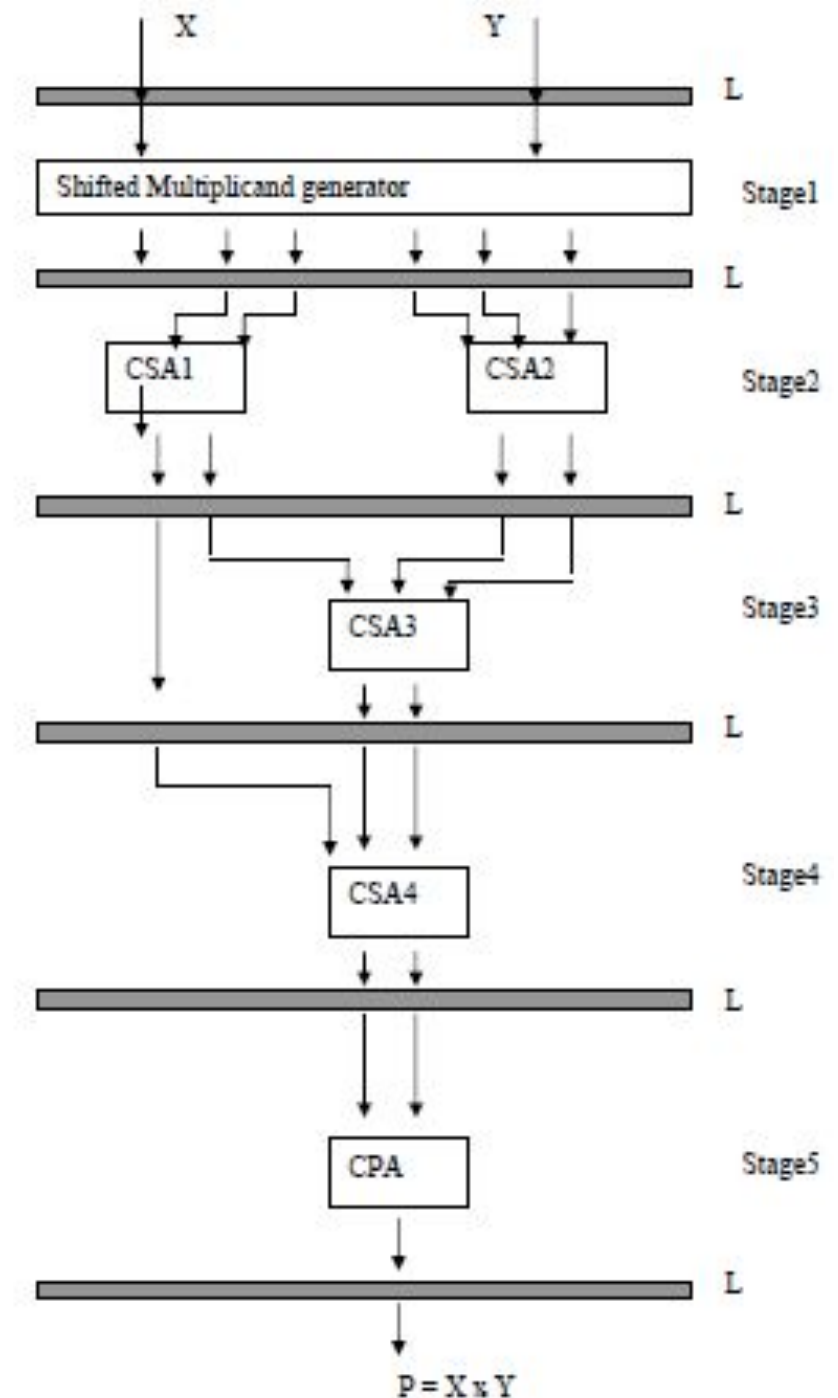
$$X_5Y_1 \ X_4Y_1 \ X_3Y_1 \ X_2Y_1 \ X_1Y_1 \ X_0Y_1 = P_2$$

$$X_5Y_2 \ X_4Y_2 \ X_3Y_2 \ X_2Y_2 \ X_1Y_2 \ X_0Y_2 = P_3$$

$$X_5Y_3 \ X_4Y_3 \ X_3Y_3 \ X_2Y_3 \ X_1Y_3 \ X_0Y_3 = P_4$$

$$X_5Y_4 \ X_4Y_4 \ X_3Y_4 \ X_2Y_4 \ X_1Y_4 \ X_0Y_4 = P_5$$

$$X_5Y_5 \ X_4Y_5 \ X_3Y_5 \ X_2Y_5 \ X_1Y_5 \ X_0Y_5 = P_6$$

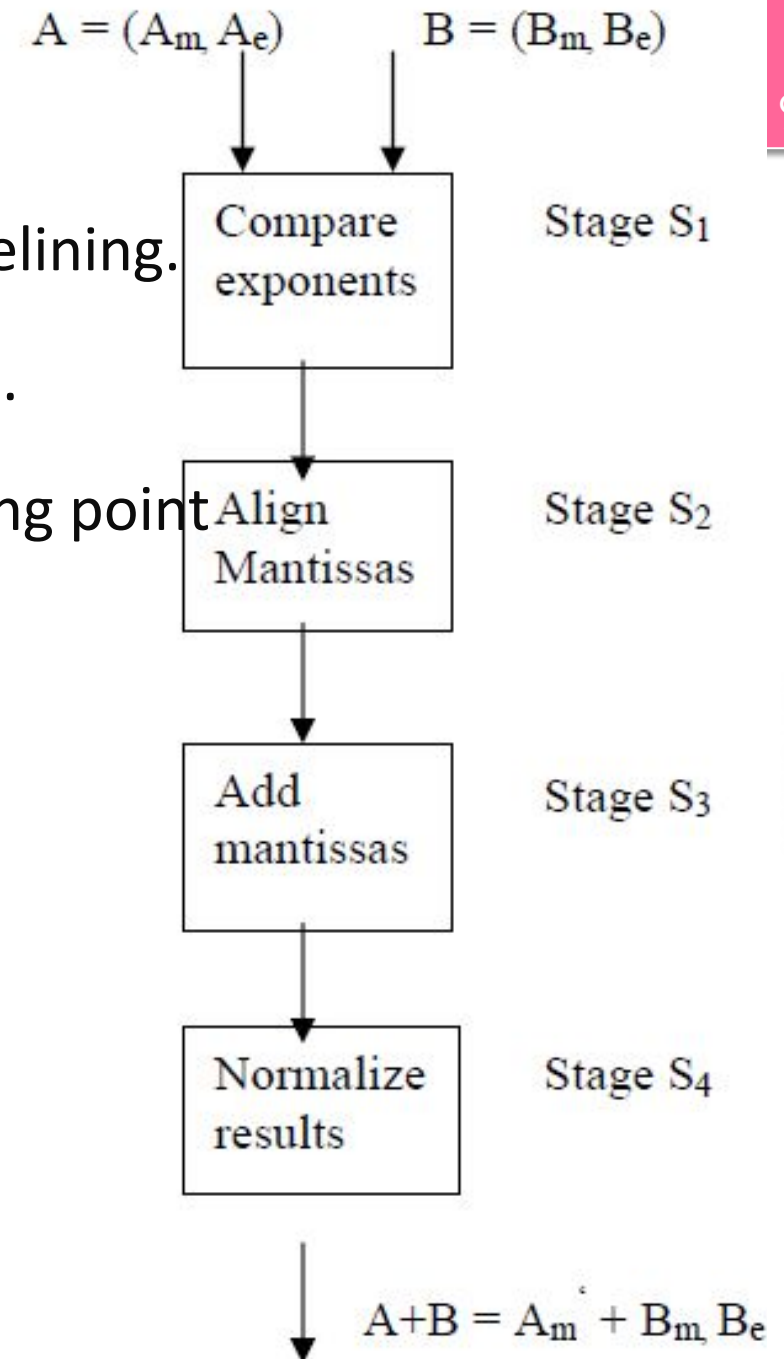



Fixed Arithmetic Pipelining- example of multiplication

- The **first stage** generates the **partial product** of the numbers, which form the six rows of shifted multiplicands.
- In the **second stage**, the six numbers are given to the two CSAs merging into four numbers.
- In the **third stage**, there is a single CSA merging the numbers into 3 numbers.
- In the **fourth stage**, there is a single number merging three numbers into 2 numbers.
- In the **fifth stage**, the last two numbers are added through a CPA to get the final product.

Floating point Arithmetic pipelines

- Floating point computations are the best candidates for pipelining.
- Consider example of **addition of two floating point numbers**.
- Following stages are identified for the addition of two floating point numbers:
 - **First stage** will compare the exponents of the two numbers.
 - **Second stage** will look for alignment of mantissas.
 - In the **third stage**, mantissas are added.
 - In the **last stage**, the result is normalized.





HAZARD DETECTION AND RESOLUTION

PIPELINE CONFLICTS:

- **Resource Conflicts:** They are caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
 - **Data Dependency:** these conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
 - **Branch Difference:** they arise from branch and other instructions that change the value of PC.
- When one of the pipeline stage is not able to complete its processing task for a given instruction in the time allotted, the pipelined operation is said to have **stalled**.
- Any condition that cause pipeline to stall is called a **hazard(unexpected negative situations)**.

Pipelining- Hazard Types

- **Data hazard:** Any condition in which either the source or destination operands of an instruction are not available at the time expected in the pipeline.
 - As a result some operation has to be delayed and the pipeline stalls.
- **Control hazards or Instruction hazards:** The pipeline may also be stalled because of a delay in the availability of an instruction.
 - For example, this may result a miss in the cache requiring the instruction to be fetched from the main memory.
- **Structural hazard** occur when there is a situation when two instructions require the use of a given hardware resource at the same time.
 - Most common case is access to memory. One instruction may need to access memory as part of execute or write stage while the other is being fetched.

Pipelining- Hazard Detection and Resolution

- Pipeline hazards are caused by **resource usage conflicts** among various instructions in the pipeline.
- Such hazards are triggered by **inter instruction dependencies** when successive instructions overlap their fetch, decode and execution through a pipeline processor
- Inter instruction dependencies may arise to prevent the **sequential data flow** in the pipeline.
 - For example an instruction may depend on the results of a previous instruction.
 - Until the completion of the previous instruction, the present instruction cannot be initiated into the pipeline.
- In other instances, two stages of a pipeline may need to update the **same memory location**.
- Hazards of this sort, if not properly detected and resolved could result in an **inter lock situation** in the pipeline or produce **unreliable results** by **overwriting**.

Pipelining- Hazard Detection and Resolution

□ There are **three classes of data dependent hazards**, according to various data update patterns:

1. **Write After Read hazards (WAR)** : may occur when J attempt to modify some data object that is read by I.
2. **Read After Write hazards (RAW)** : hazard between the two instructions I and J may occur when they attempt to read some data object that has been modified by I.
3. **Write After Write hazards (WAW)** : may occur if both I and J attempt to modify the same data object.
4. **Read After Read** does not pose a problem because nothing is changed

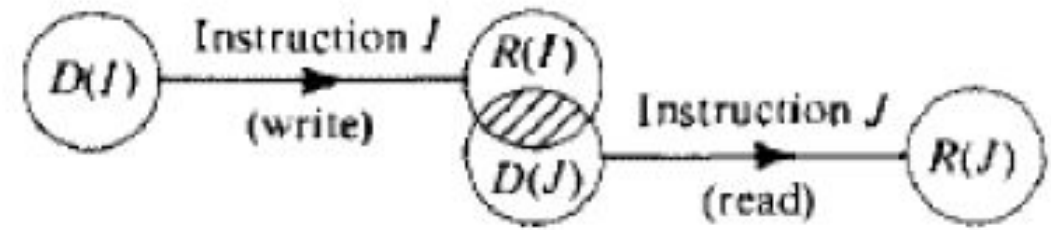
Pipelining- Hazard Detection and Resolution

- We use **Resource Object** to refer to **working registers**, **memory locations** and **special flags**.
- The contents of these resource objects are called **data objects**.
- Each instruction can be considered a mapping from a set of data objects to a set of data objects.
- The **Domain $D(I)$** of an instruction I is a **set of resource objects** whose data objects may affect the execution of instruction I .
- The **Range $R(I)$** of an instruction is the **set of resource objects** whose data objects may be **modified** by the execution of instruction I .
- Obviously, the operands to be used in an instruction execution are retrieved (**read**) **from its domain** and the results will be stored (**written**) **in its range**.

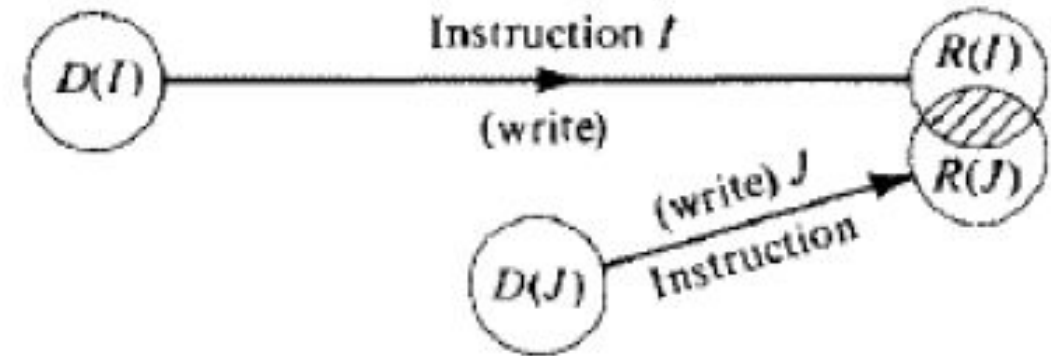
Pipelining- Hazard Detection and Resolution

26

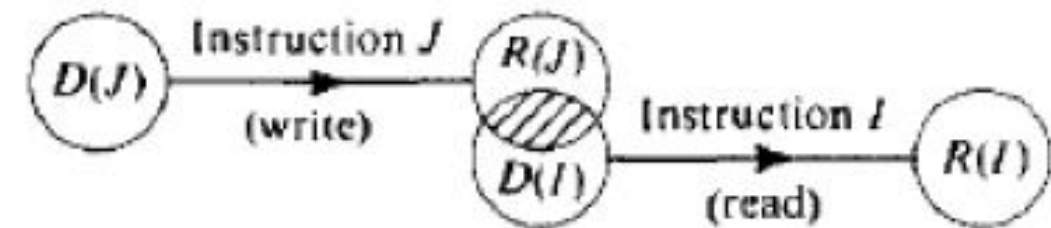
$R(I) \cap D(J) \neq \phi$ for RAW
 $R(I) \cap R(J) \neq \phi$ for WAW
 $D(I) \cap R(J) \neq \phi$ for WAR



(a) RAW hazard



(b) WAW hazard



(c) WAR hazard

Pipelining- Hazard Detection and Resolution

- Once the hazard is detected, the system should resolve the **interlock situation**
 - A straightforward approach is to **stop the pipe** and to **suspend the execution** of the coming instructions until instruction I has passed the point of resource conflict.
 - A more sophisticated approach is to suspend only next instruction J and continue the flow of instruction down the pipe
 - Multi level hazard detection may be encountered, requiring more complex control mechanisms to resolve a stack of hazards

Pipelining- Hazard Detection and Resolution

- In order to avoid RAW hazards, IBM engineers developed a **short circuiting approach** which gives a **copy of the data object** to be written **directly to the instruction waiting** to read the data.
- This concept was generalized into a technique known as **data forwarding**, which **forward multiple copies of the data** to as many waiting instructions as may wish to read it.
 - A **data forwarding chain** can be established in some cases.
- The **internal forwarding** and **register-tagging** techniques are helpful in resolving logic hazards in pipelines