Pipelining Hazards

Pipelining is not suitable for all kinds of instructions. When some instructions are executed in pipelining they can stall the pipeline or flush it totally. This type of problems caused during pipelining is called Pipelining Hazards.

The three different types of hazards in computer architecture are:

Structural Hazards :

Structural Hazards arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU.

Data Hazards :

Data Hazards occur when an instruction depends on the result of previous instruction and that result of instruction has not yet been computed. whenever two different instructions use the same storage. the location must appear as if it is executedinsequentialorder.

Therearefourtypesofdatadependencies:

ReadafterWrite(RAW)

WriteafterRead(WAR)

WriteafterWrite(WAW)

Read after Read (RAR)

Control Hazards :

Control Hazards occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc. On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline. Due to this, unwanted instructions are fed to the pipeline.

Performance Evolution Factor for Pipelining

- **Latency**:

  Latency is the time taken for a single instruction to complete its execution. A lower latency indicates better performance.
- **Efficiency**:

  Pipeline efficiency measures how effectively the pipeline stages are utilized. It is calculated by dividing the total time spent on useful work by the total time taken for the entire pipeline.
- **Throughput** :

  Throughput refers to the number of instructions completed per unit of time. In a pipeline processor, a higher throughput indicates better performance.

Benefits of Pipelining

- Increases instruction throughput.
- Better utilization of processor components.
- Efficient for repetitive tasks or streaming data.

Disadvantages of Pipelining

- Designing of the pipelined processor is complex.
- The problem of hazard for branch instructions increases with the longer pipeline.
- It is difficult to predict the throughput of a pipelined processor.

Dependencies in a pipelined processor There are mainly three types of dependencies possible in a pipelined processor. These are : 1) Structural Dependency 2) Control Dependency 3) Data Dependency These dependencies may introduce stalls in the pipeline. **Stall :** A stall is a cycle in the pipeline without new input. **Structural dependency** This dependency arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU. Example:

Instruction / Cycle	**1**	**2**	**3**	**4**	**5**
I_1	IF(Mem)	ID	EX	Mem	
I_2		IF(Mem)	ID	EX	
I_3			IF(Mem)	ID	EX
I_4				IF(Mem)	ID

In the above scenario, in cycle 4, instructions I_1 and I_4 are trying to access same resource (Memory) which introduces a resource conflict. To avoid this problem, we have to keep the instruction on wait until the required resource (memory in our case) becomes available. This wait will introduce stalls in the pipeline as shown below:

Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
I_1	IF(Mem)	ID	EX	Mem	WB			
I_2		IF(Mem)	ID	EX	Mem	WB		
I_3			IF(Mem)	ID	EX	Mem	WB	
I_4				-	-	-	IF(Mem)	

Solution for structural dependency To minimize structural dependency stalls in the pipeline, we use a hardware mechanism called Renaming.

Renaming : According to renaming, we divide the memory into two independent modules used to store the instruction and data separately called Code memory(CM) and Data memory(DM) respectively. CM will contain all the instructions and DM will contain all the operands that are required for the instructions.

Instruction/ Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**
I_1	IF(CM)	ID	EX	DM	WB		
I_2		IF(CM)	ID	EX	DM	WB	
I_3			IF(CM)	ID	EX	DM	WB
I_4				IF(CM)	ID	EX	DM
I_5					IF(CM)	ID	EX
I_6						IF(CM)	ID
I_7							IF(CM)

Control Dependency (Branch Hazards) This type of dependency occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc. On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline. Due to this, unwanted instructions are fed to the pipeline. Consider the following sequence of instructions in the program: 100: I_1 101: I_2 (JMP 250) 102: I_3 . . 250: BI_1 Expected output: I_1 -> I_2 -> BI_1 NOTE: Generally, the target address of the JMP instruction is known after ID stage only.

Instruction/ Cycle	**1**	**2**	**3**	**4**	**5**	**6**
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC:250)	EX	Mem	WB
I_3			IF	ID	EX	Mem
BI_1				IF	ID	EX

Output Sequence: I_1 -> I_2 -> I_3 -> BI_1 So, the output sequence is not equal to the expected output, that means the pipeline is not implemented correctly. To correct the above problem we need to stop the Instruction fetch until we get target address of branch instruction. This can be implemented by introducing delay slot until we get the target address.

Instruction/ Cycle	**1**	**2**	**3**	**4**	**5**	**6**
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC:250)	EX	Mem	WB
Delay	-	-	-	-	-	-
BI_1				IF	ID	EX

Output Sequence: I_1 -> I_2 -> Delay (Stall) -> BI_1 As the delay slot performs no operation, this output sequence is equal to the expected output sequence. But this slot introduces stall in the pipeline. **Solution for Control dependency** Branch Prediction is the method through which stalls due to control dependency can be eliminated. In this at 1st stage prediction is done about which branch will be taken.For branch prediction Branch penalty is zero. **Branch penalty :** The number of stalls introduced during the branch operations in the pipelined processor is known as branch penalty

Example: Let there be two instructions I1 and I2 such that: I1 : ADD R1, R2, R3 I2 : SUB R4, R1, R2 When the above instructions are executed in a pipelined processor, then data dependency condition will occur, which means that I_2 tries to read the data before I_1 writes it, therefore, I_2 incorrectly gets the old value from I_1.

Instruction / Cycle	**1**	**2**	**3**	**4**
I_1	IF	ID	EX	DM
I_2		IF	ID(Old value)	EX

To minimize data dependency stalls in the pipeline, **operand forwarding** is used. **Operand Forwarding :** In operand forwarding, we use the interface registers present between the stages to hold intermediate output so that dependent instruction can access new value from the interface register directly. Operand Forwarding can avoid stalls only if the dependent instructions are ALU type instructions. Considering the same example: I1 : ADD R1, R2, R3 I2 : SUB R4, R1, R2

Instruction / Cycle	**1**	**2**	**3**	**4**
I_1	IF	ID	EX	DM
I_2		IF	ID	EX

Data Hazards Data hazards occur when instructions that exhibit data dependence, modify data in different stages of a pipeline. Hazard cause delays in the pipeline. There are mainly three types of data hazards: 1) RAW (Read after Write) [Flow/True data dependency] 2) WAR (Write after Read) [Anti-Data dependency] 3) WAW (Write after Write) [Output data dependency] Let there be two instructions I and J, such that J follow I. Then,

- Instruction depend on result of prior instruction still in the pipeline.
- It can occur among the operands in the instruction at the pipeline stages.
- It occur when instructions read or write registers that are used by other instructions.
- RAW hazard occurs when instruction J tries to read data before instruction I writes it. Eg: I: R2 <- R1 + R3 J: R4 <- R2 + R3
- WAR hazard occurs when instruction J tries to write data before instruction I reads it. Eg: I: R2 <- R1 + R3 J: R3 <- R4 + R5
- WAW hazard occurs when instruction J tries to write output before instruction I writes it. Eg: I: R2 <- R1 + R3 J: R2 <- R4 + R5