**Classification of pipeline processor**

The pipeline processors were classified based on the levels of processing by Handler in 1977.

Given below are the classification of pipeline processor by given by Handler:

- **Arithmetic pipeline**
- **Processor pipeline**
- **Instruction pipeline**

**Arithmetic pipeline**

An arithmetic pipeline generally breaks an arithmetic operation into multiple arithmetic steps.

So in arithmetic pipeline, an arithmetic operation like multiplication, addition, etc. can be divided into series of steps that can be executed one by one in stages in Arithmetic Logic Unit (ALU).

**Example of Arithmetic pipeline**

Listed below are examples of arithmetic pipeline processor:

8 stage pipeline used in TI-ASC

4 stage pipeline used in Star-100

**Processor pipeline**

In processor pipeline processing of the same data stream is done by a cascade of processors.

In this each cascade of processor is assigned and process a specific task.

The processors are pipelined to process the same data stream. The data stream is processed by the first processor and the result is stored in the memory block. The result in the memory block is accessed by the second processor. The second processor reprocesses the result obtained by the first processor and the passes the refined result to the third processor and so on.

There is no practical example found for processor pipeline.

**Instruction pipeline**

In instruction pipeline processor, the execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode and operand fetch of subsequent instructions.

**Example of Instruction pipeline**

All high-performance computers nowadays are equipped with instruction pipeline processor.

## Collisions in the pipeline

We can’t really keep the example instruction pipeline full if we have a single cache for accessing memory.

For example, we would be trying to:

¨ store results of instruction 1 while we are ¨ fetching an operand for instruction 3 and ¨ fetching instruction 6 all at the same time. These are called collisions in the pipeline

**Reservation Table**

A reservation table is a way of representing the task flow pattern of a pipelined system. A reservation table has several rows and columns. Each row of the reservation table represents one resource of the pipeline and each column represents one time-slice of the pipeline. For example, if in a pipelined system, there are four resources and five time-slices, then, the reservation table will have four rows and five columns. All the elements of the table are either 0 or 1. If one resource (say, resource i) is used in a time-slice (say time-slice j), then the (i,j)-th element of the table will have the entry 1. On the other hand, if a resource is not used in a particular time-slice, then that entry of the table will have the value 0.

**An Analysis Example of Reservation Table**

Suppose that we have 4 resources and 6 time-slices and the usage of resources is as follows :

1. resource 1 is used in time-slices 1, 3, 6.
2. resource 2 is used in time-slice 2.
3. resource 3 is used in time-slices 2, 4.
4. resource 4 is used in time-slice 5.

The corresponding reservation table will be as follows :

| index | time-1 | time-2 | time-3 | time-4 | time-5 | time-6 |
|---|---|---|---|---|---|---|
| resource 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| resource 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| resource 3 | 0 | 1 | 0 | 1 | 0 | 0 |
| resource 4 | 0 | 0 | 0 | 0 | 1 | 0 |

Often to make the table look simpler, the 0 entries are represented by blank and 1 entries are represented by a 'X'. We have followed this second approach when we display the reservation table. The above table will look now like

| index | time-1 | time-2 | time-3 | time-4 | time-5 | time-6 |
|---|---|---|---|---|---|---|
| resource 1 | X |  | X |  |  | X |
| resource 2 |  | X |  |  |  |  |
| resource 3 |  | X |  | X |  |  |
| resource 4 |  |  |  |  | X |  |

The reservation table A reservation table is a very useful tool for analyzing pipelines.

This would indicate that there are no collisions in the pipeline, and that we could initiate a new instruction every  clock periods.

## Structural Hazards

- Suppose the IF (instruction fetch) stage and the OF (operand fetch) stages both need to access main memory for their data.
- Suppose we only have one data path to main memory.
- We will not be able to execute an IF and an OF at the same time.

|  | Time |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 |
| I1 | IF | ID | OF | EX | WB |  |  |  |
| I2 |  | IF | ID | OF | EX | WB |  |  |
| I3 |  |  | IF | ID | OF | EX | WB |  |

---

## Structural Hazard Remedy: Duplicate Hardware

- One solution is to add redundant hardware to avoid these hazards.
- Decision depends on the design costs, transitor constraints and the performance increase it will yield.

---

## Structural Hazard Remedy: Stalling

- Another solution is to **stall** the pipeline, but this will reduce the overall processor performance.

|  | Time |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 |
| I1 | IF | ID | OF | EX | WB |  |  |  |
| I2 |  | IF | ID | OF | EX | WB |  |  |
| I3 |  |  | -- | IF | ID | OF | EX | WB |

- In this case, stalling for one stage results in another conflict so we must stall for two stages.

|  | Time |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 |
| I1 | IF | ID | OF | EX | WB |  |  |  |
| I2 |  | IF | ID | OF | EX | WB |  |  |
| I3 |  |  | -- | -- | IF | ID | OF | EX |

# Data Hazards

Consider the following 2-instruction code using our 5 stage pipeline:

I1: R3 := R1 + R2

I2: R4 := R3 + 10

|  | Time |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 |
| I1 | IF | ID | OF | EX | WB |  |  |  |
| I2 |  | IF | ID | OF | EX | WB |  |  |

- Instruction I2 needs the data from R3 at time 't4' (operand fetch).
- However, I1 hasn't finished yet, so R3 value has not been written back and will not be available until after time 't5'.

Three classes of data dependencies hazards, according to various data update patterns:

**1)write after read (WAR)**

Example: Add R1, R2, R4
Sub R2, R3, R5

WAR (Write After Read)

Arises from an anti dependence

Cannot occur in most static issue pipelines

Occurs either when there are early writes and late reads, or when instructions are re-ordered

There is no actual data transfer. It is the same name that causes the problem

Considering two instructions i and j, instruction j should write after instruction i has read the data.

**2)read after write (RAW)**

Corresponds to a true data dependence

Program order must be preserved

This hazard results from an actual need for communication

Considering two instructions i and j, instruction j reads the data before i writes it

i:  ADD R1, R2, R3

j:  SUB R4, R1, R3

3)**write after write (WAW)** Corresponds to an output dependence

Occurs when there are multiple writes or a short integer pipeline and a longer floating-point pipeline or when an instruction proceeds when a previous instruction is stalled WAW (write after write)

This is caused by a name dependence. There is no actual data transfer. It is the same name that causes the problem

Considering two instructions i and j, instruction j should write after instruction i has written the data

i:  SUB R1, R4, R3

j:  ADD R1, R2, R3

Instruction i has to modify register r1 first, and then j has to modify it. Otherwise, there is a WAW hazard. There is a problem because of R1. If some other register had been used, there will not be a problem

**Solution** is register renaming, that is, use some other register. The hardware can do the renaming or the compiler can do the renaming

---

## Data Hazard Remedy: Stalling

| | Time | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 |
| I1 | IF | ID | OF | EX | WB | | | |
| I2 | | IF | ID | -- | -- | OF | EX | WB |

- slows down execution

---

## Data Hazard Remedy: Data Forwarding

| | Time | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 |
| I1 | IF | ID | OF | EX | WB | | | |
| I2 | | IF | ID | -- | OF | EX | WB | |

- requires extra hardware/more complex hardware design