**Adv CA Mid Term Practce:**

Problem 1 :

1. Useful Work per stage = useful work / #stages.

Minimum clock period = total useful work + total latch latency.

Max frequency = 1/ f clock period

1. Speed up = Performance of 14 stage pipeline / performance of 5 stage pipeline.

Problem 2:

1. 6 : All instructions that are **fetched after the mispredicted branch** and have **entered the pipeline before the Memory stage** must be flushed.

**20% of instructions are branches** → Branch frequency = 0.20

**90% prediction accuracy** → 10% of branches are mispredicted

**Each mispredicted branch flushes 6 instructions** (from part i)

**No other hazards**

**Base CPI = 1**

A screenshot of a computer

AI-generated content may be incorrect.

Problem 3:

1. Explain how aliasing in a branch predictor can cause mispredictions

Aliasing in a branch predictor occurs when multiple branches map to the same entry in the prediction table. This happens due to limited table size or overlapping index values (e.g., same PC bits or global history).

When aliasing occurs, different branches with different behaviors update the same table entry, causing conflicting updates. As a result, the predictor may learn the behavior of one branch and then get overwritten by another, leading to mispredictions, even if both branches are individually predictable.

A close up of a white box

AI-generated content may be incorrect.

A black text on a white background

AI-generated content may be incorrect.

A white paper with black text

AI-generated content may be incorrect.

**🧠 Let's go step by step to solve this problem.**

We are given the following delays for each stage of a **pipelined datapath** and **latency for pipeline latches**:

**Given Information:**

* **Pipeline Stage Delays**:
  + Fetch: **150 ps**
  + Delay: **75 ps**
  + Decode: **100 ps**
  + Delay2: **75 ps**
  + Execute: **125 ps**
  + Delay3: **75 ps**
  + Memory/Branch: **150 ps**
  + Delay4: **75 ps**
  + Write Back: **75 ps**
* **Latency of Pipeline Latches**: **25 ps**

**Steps to Calculate the Minimum Clock Period:**

1. **Calculate the Delay for Each Stage**:
   * The clock period must be at least as long as the stage with the **longest delay**, so we need to take the maximum delay from the stages, including the pipeline latch latency.
2. **Add the Latency of Pipeline Latches**:
   * Pipeline latches add a latency of **25 ps** to each stage. So, we will add **25 ps** to the delay of each stage.
3. **Find the Maximum Delay**:
   * The stage with the maximum delay will set the minimum clock period because the clock must wait for the longest stage to finish before the next stage can start.

**Step-by-Step Calculation:**

Let's compute the total delay for each stage, including the latch latency:

| **Stage** | **Delay (ps)** | **Latch Latency (ps)** | **Total Delay (ps)** |
| --- | --- | --- | --- |
| Fetch | 150 | 25 | 175 |
| Delay | 75 | 25 | 100 |
| Decode | 100 | 25 | 125 |
| Delay2 | 75 | 25 | 100 |
| Execute | 125 | 25 | 150 |
| Delay3 | 75 | 25 | 100 |
| Memory/Branch | 150 | 25 | 175 |
| Delay4 | 75 | 25 | 100 |
| Write Back | 75 | 25 | 100 |

**Determine Minimum Clock Period:**

The minimum clock period is determined by the stage with the **maximum total delay**. From the table above:

* The maximum delay is **175 ps**, which occurs in both **Fetch** and **Memory/Branch** stages.

Thus, the **minimum clock period** is **175 ps**.

**A screenshot of a computer

AI-generated content may be incorrect.**

Let me know if you'd like to dive deeper into any of the steps!

**Question: Explain the need for branch prediction in modern processors?**

Branch prediction is crucial in modern processors because it minimizes pipeline stalls caused by branch instructions. By predicting the outcome of branches (whether they are taken or not), the processor can continue executing instructions without waiting for the branch decision, improving throughput and efficiency. Accurate predictions reduce the need for pipeline flushing, saving cycles and boosting performance.

**Question: Explain the Coherence problem in shared memory multiprocessor systems with  
local caches.**

The **coherence problem** in shared memory multiprocessor systems arises when multiple processors have local caches that store copies of the same memory location. The issue occurs when one processor updates a memory location, but other processors' caches don't reflect the change, leading to **inconsistent data**. Cache coherence protocols (e.g., MESI) are used to ensure that all caches maintain consistent data by synchronizing updates across processors.

A close up of a text

AI-generated content may be incorrect.

Qstn : Explain how aliasing in a branch predictor can cause mispredictions?

Aliasing in a branch predictor happens when multiple branches map to the same entry in the prediction table, causing the predictor to confuse one branch with another. This leads to mispredictions, as the predictor may incorrectly predict the outcome of a branch, resulting in pipeline stalls and wasted cycles.