Computer Architecture week 4

Khaula Molapo

1.

A common hazard scenario in gaming consoles is overheating during extended gameplay. Modern consoles generate significant heat due to powerful CPUs and GPUs. If ventilation is blocked or the internal cooling system fails, components can reach unsafe temperatures. Overheating can cause sudden shutdowns, data corruption, or permanent hardware damage. For example, a gamer playing for several hours in a closed room without proper airflow may notice the console fans running loudly, warning lights blinking, or the system automatically powering down. This hazard can also affect the surrounding environment, as hot surfaces may cause burns if touched. Manufacturers implement thermal sensors and automatic fan control to reduce the risk, but users must ensure proper placement and avoid obstructing air vents. Overheating hazards highlight the importance of monitoring temperature, maintaining hardware, and using consoles responsibly to prevent safety risks and preserve the lifespan of gaming devices.

2.

Branch prediction is a CPU technique that improves performance by guessing the outcome of conditional operations. When a program contains decision points like if-else statements or loops, the CPU may stall while waiting to determine the correct path. Branch prediction predicts which path will be taken and preloads instructions accordingly. If the prediction is correct, the pipeline continues without interruption, reducing delays. Incorrect predictions require the pipeline to flush and reload, causing some performance loss. Branch prediction is essential for minimizing pipeline hazards in modern processors, ensuring faster execution, better instruction flow, and improved efficiency in applications such as gaming, scientific computing, and multitasking environments.

3.

Reflection: This simulation models pipeline hazards by checking if an instruction is already in the pipeline. Writing it in Visual Studio Code helped me understand data hazards and instruction dependencies. Each instruction moves through stages, and conflicts simulate hazards. The program demonstrates how hazards can slow down execution and why processors

implement techniques like forwarding or stalling. Testing this simulation gave me a hands-on view of CPU pipeline challenges, showing the importance of hazard detection and mitigation. Connecting theory with practical coding clarified how processors maintain instruction flow while avoiding errors, making it easier to grasp modern CPU design principles and performance optimization strategies.

4.

The pipeline hazard diagram illustrates instruction flow in a CPU pipeline and potential hazards. It shows stages like fetch, decode, execute, memory, and write-back. Data hazards occur when instructions depend on previous results, control hazards happen during branches, and structural hazards appear when resources are limited. Arrows indicate instruction movement, and highlights show where conflicts may occur. The diagram emphasizes hazard detection, stalling, and forwarding techniques used to maintain performance. This visual representation makes it easier to understand instruction dependencies, timing conflicts, and how modern CPUs handle multiple instructions efficiently. Pipeline diagrams are essential for studying CPU architecture and optimization.