Title:

Analysis of Pipeline Execution in Floating-Point Instructions

---

1. Aim

To analyse the execution of floating-point instructions in a pipelined processor, understand different pipeline stages, and study the impact of data hazards, stalls, and forwarding mechanisms in enhancing instruction throughput.

---

2. Introduction

In modern processors, instruction pipelining is used to enhance performance by overlapping execution stages of multiple instructions. This report analyses a simulated pipeline execution for floating-point instructions. The key aspects studied include:

- **Instruction execution in various pipeline stages.**
- **Data hazards and their impact on performance.**
- **The role of data forwarding in minimizing stalls.**
- **Execution latency of different floating-point instructions.**
- **Behaviour of instructions under different dependency scenarios.**
- **The importance of instruction ordering in avoiding performance loss.**

---

3. Methodology

The simulation was performed using an online pipeline simulator (UMass Cache & Pipeline Simulator). The following steps were taken:

- **A custom sequence of floating-point instructions was input into the simulator.**
- **Execution latencies were set for each instruction type (Addition, Subtraction, Division).**
- **Forwarding was enabled to observe how data is passed between stages.**
- **Stalls due to RAW (Read After Write) dependencies were analysed.**
- **The pipeline diagram output was examined to identify delays and overlaps.**
- **Different configurations were tested (with and without forwarding) to compare performance.**

4. Observations

Instructions Executed:

1. **fp_ld F6, Offset(R2) – Loads floating-point register F6 from memory.**
2. **fp_ld F2, Offset(R3) – Loads floating-point register F2.**
3. **fp_div F1, F2, F4 – Performs floating-point division.**
4. **fp_sub F8, F2, F6 – Performs floating-point subtraction.**
5. **fp_div F10, F1, F6 – Second division dependent on result of previous instruction.**
6. **fp_add F6, F8, F2 – Performs floating-point addition.**

Key Observations:

	Instruction	1	2	3	4	5	6	7	8	9	10	11
0	fp_ld (F6, Offset, R2)	IF	ID	EX	MEM	WB						
1	fp_ld (F2, Offset, R3)		IF	ID	EX	MEM	WB					
2	fp_div (F1, F2, F4)			IF	ID	S	/ (p)	MEM	WB			
3	fp_sub (F8, F2, F6)				IF	S	ID	+|- (f)	MEM	WB		
4	fp_div (F10, F1, F6)						IF	ID	/ (p)	MEM	WB	
5	fp_add (F6, F8, F2)							IF	ID	+|- (f)	MEM	WB

Step | Execute All Instructions

- **Pipeline Stages: Instructions moved through 5 main stages: IF → ID → EX → MEM → WB.**
- **Data Hazards: Instructions like fp_div F10, F1, F6 depended on earlier results (F1, F6) which caused pipeline stalls.**
- **Forwarding Enabled: Forwarding helped minimize the number of stall cycles by passing results directly from EX or MEM to dependent instructions.**
- **Execution Latency:**
  - **Division operations (fp_div) took multiple EX cycles (deep pipelines).**
  - **Arithmetic operations (fp_add, fp_sub) also required more than one cycle depending on pipeline settings.**
- **Instruction Dependencies:**
  - **RAW hazards were identified between fp_div → fp_div, fp_ld → fp_sub, and fp_sub → fp_add.**

- **Stalls Indicated in Red (S): These represent pipeline stalls where instructions must wait for operand availability.**

---

5. Conclusion

- **Pipelining significantly boosts CPU throughput by allowing multiple instructions to overlap in execution.**
- **Data hazards (especially RAW) are a major cause of performance degradation and require techniques like stalling or forwarding to resolve.**
- **Data forwarding is effective in reducing pipeline stalls, but it cannot eliminate all hazards, especially in back-to-back dependencies or when instructions require results from multi-cycle operations.**
- **Floating-point operations are more complex, often requiring multiple cycles to complete (especially division).**
- **Simulator tools are useful in visualizing pipeline execution and understanding the internal working of processors.**
- **Proper instruction scheduling and reordering can help reduce stalls and enhance overall performance.**
- **This analysis provides a foundational understanding of how modern CPUs handle instruction-level parallelism and the challenges they face due to data and control hazards.**