Pipelining.

- Each stage has to be independentlly clk and operate.
- Thus, every stage needs to connect to a clk.

Except the memory that runs on different clock.

- B/w each stage of pipline, there is a latch to store~~d~~ the data.
- Speed up for free

Pipelining Cons:

Complexity - The more piplining the more prone to error.

- The edgy cases don't work well
- Needs to have a better error handler.

None-determinism

- Can't gauruntced the actual execution time for each instr. (Though you can rely on the Avg. time)
  - → May to consider pipelining depending on use cases.
- Can't be really use for Syncing in a high freq clk.
  - → Propagation delay at phisical level ∴ Best used in Low freq clk μController
- Can't branch well — Severe performanc penalty.
  - old instr
  - → Need to flush whatev in the dest. and reload it again. later.
  - → Then you have N cycles for 1 instr. Once the pipeline is full again, each clk is an instr. (You have introduced Bubble in the pipeline).

Side Track note.

GPU Computing: - Specialized optimization for single float comp
- Highly parallel

- Very deep pipelining is used b/c there is not much branching in Graphic Computation Algorithm.

(Con of Pipeline)

Instruction Ordering:

- Pipeline Harzard: Write after Read
  - → Intel has out-of-order execution.
    - └ solved by adding an arbitrary data-indep and execute it b/w the same register read/write issue.
  - → You don't want to read ar none updated data.

Note: If you want to use pipeline avoid branching. Branching is NOT Jumping (there is a comparision involved in Branching while Jump doesn't).

So, There is the thing call Branch Prediction. it predicts what branch it will take.

Loop-unrolling is a concept that the compiler knows how many times you are going to run the section of code. (i.e. for-loop). The compiler will recognize it copy and paste the block of code (in asm) that many time and get rid of the branching. Thus, yield a better performance (but bigger asm code).