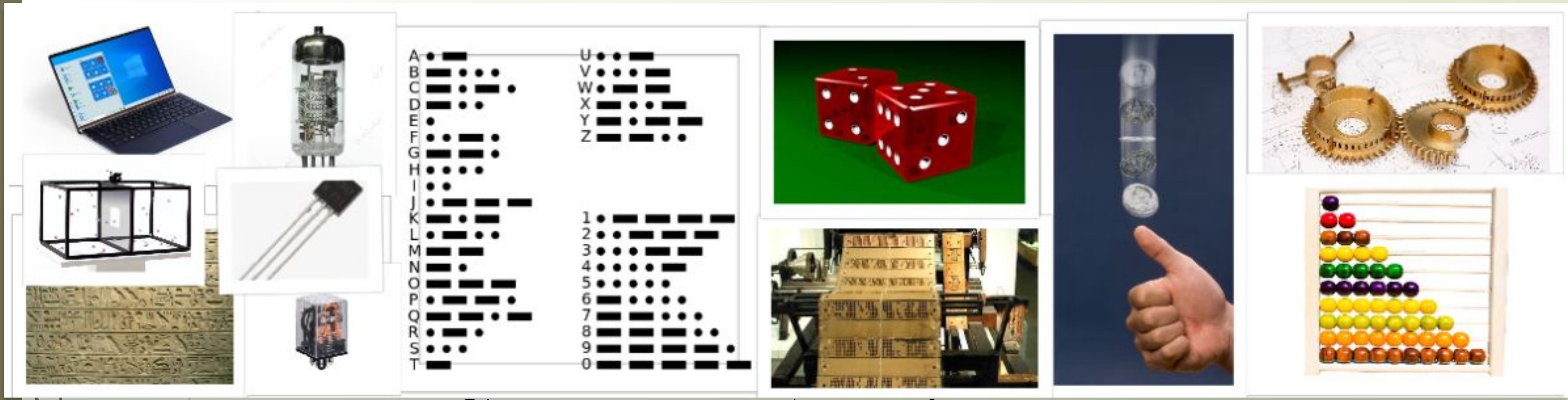


Computer Architecture (CS F342)

Design, Analysis, Execution and Optimization of Instructions

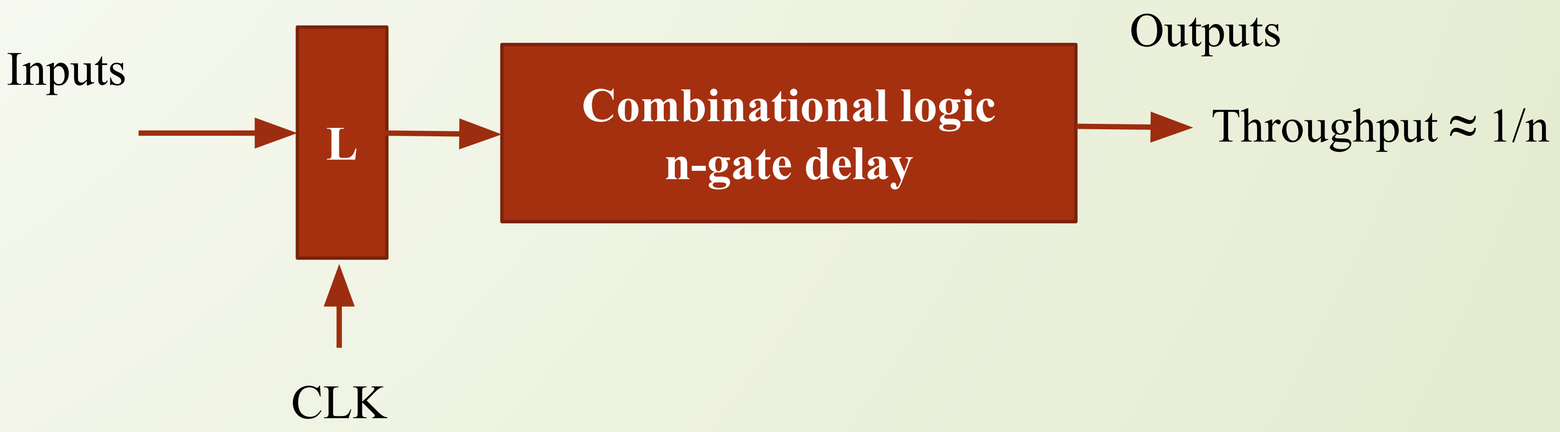
Fundamentals of Pipelined-based Design Methodology

Problems of Multi-cycle Processor

- The fundamental problem
  - Split the slowest instruction, *lw*, 5-steps
  - Processor’s clock cycle time does not improve 5-times
- The steps take unequal length of time
- 5-non-architectural registers and a additional multiplexer

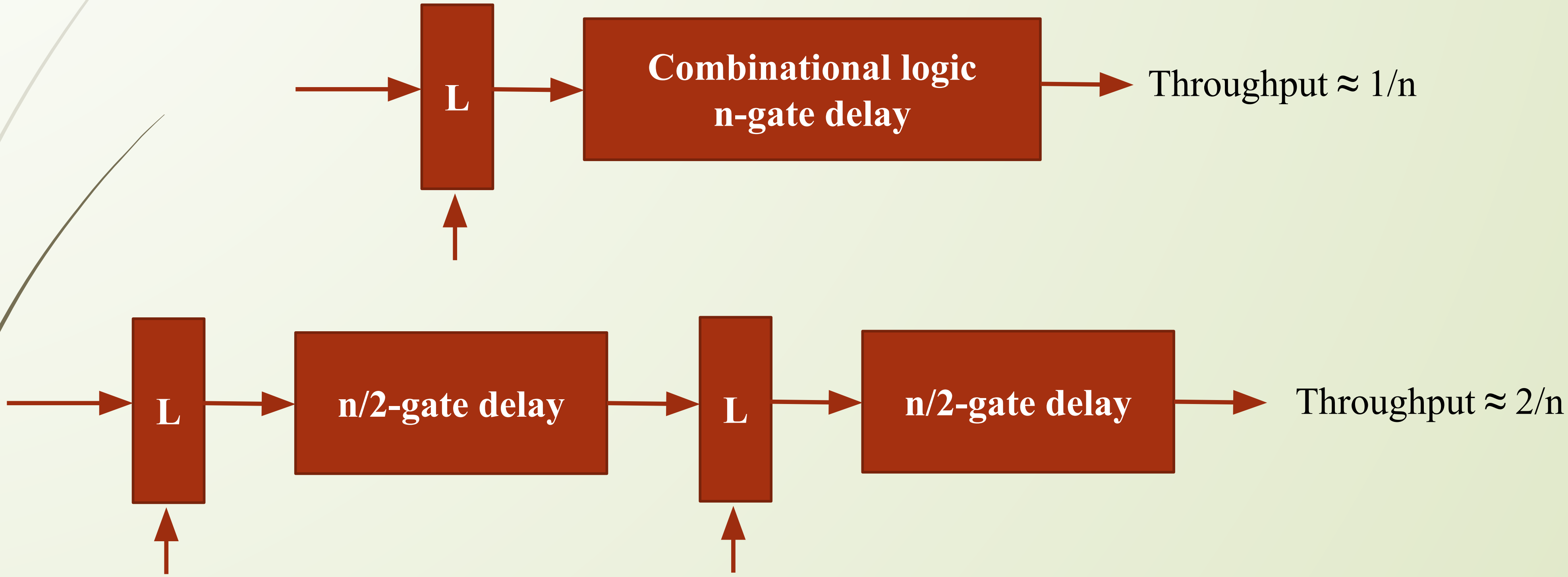
How to improve the processor's performance?

- Measure throughput (output/unit-time)
- For example



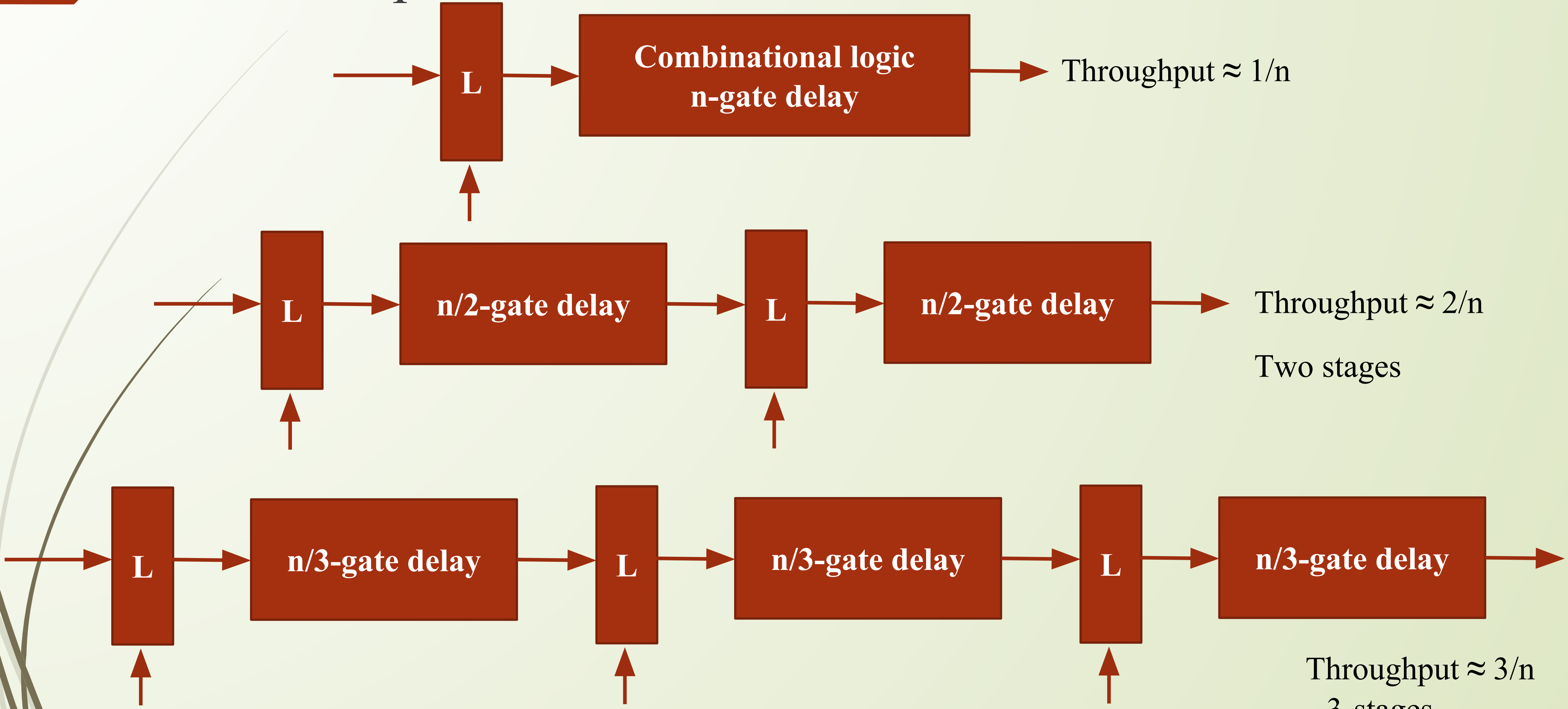
How to improve the processor's performance?

- For example



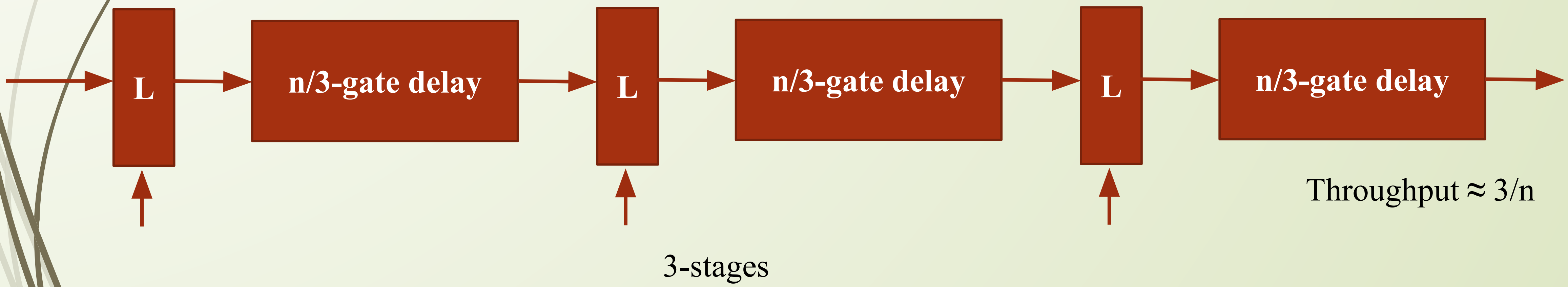
How to improve the processor's performance?

- For example



Pipelined-based Design Methodology

- k-fold increase in throughput
  - Increase in performance
    - Partitioning the logics
    - Adding new buffer
  - **Inputs are overlapped in execution**



Limitations of Pipelined-based Methodology

- Assumed inter-stage buffers does not introduce additional delay
- Increase in performance as the stages increase
- What if the stages increase to infinite?

Limitations of Pipelined-based Methodology

- What if the stages increase to infinite?
- Constrains
  - Clocking
  - Physical limitation on partitioning the logics
- Cost

Minimum clock period in Pipeline-based Systems

- Pipelined-based design
  - Combinational logic (F)
  - Latch (L)
- Max. propagation delay in F: T_M
- Min. propagation delay in F: T_m
- Proper latching delay: T_L

Minimum clock period in Pipeline-based Systems

- Consider the 2-scenarios
- Case-1:
  - Inputs x_1 applied at the stage at time T_1
  - Outputs of F must be valid at $T_1 + T_M$
  - Latching at L of the outputs must be valid until: $T_1 + T_M + T_L$

Minimum clock period in Pipeline-based Systems

- Case-2:
  - Inputs x_2 applied at the stage at time T_2
  - Effect of the outputs can be found at least at $T_2 + T_m$
  - Condition of 2-nd set of signals does not overrun the 1-st set: $T_2 + T_m > T_1 + T_M + T_L$
- Clock period (T): $T_2 - T_1 > T_M - T_m + T_L$
- Max. clocking rate cannot exceed 1/T

Minimum clock period in Pipeline-based Systems

- Clock period has two parts
  - T_M-T_m
  - T_L
- T_M-$T_m \approx 0$ How?
- T_L:
  - feedback loop and stabilizing of the signal
  - worst-case clock skew

Tradeoff between Cost and Performance

- Cost of non-pipelined design: G
  - Gate count
- Cost of adding a latch: L
- Cost of k-stages pipelined design (C): G + k * L
- Cost of pipeline design increases linearly w.r.t depth of pipeline

Tradeoff between Cost and Performance

- The latency in the non-pipeline design: T
- Performance or throughput: 1/T
- Throughput of pipelined design (P): $1/(T/k + S)$
- The additional delay S because of latches
- P is a non-linear function of k

Tradeoff between Cost and Performance

- Cost/performance ratio

$$\frac{C}{P} = \frac{G + k * L}{\frac{1}{(\frac{T}{k} + S)}}$$

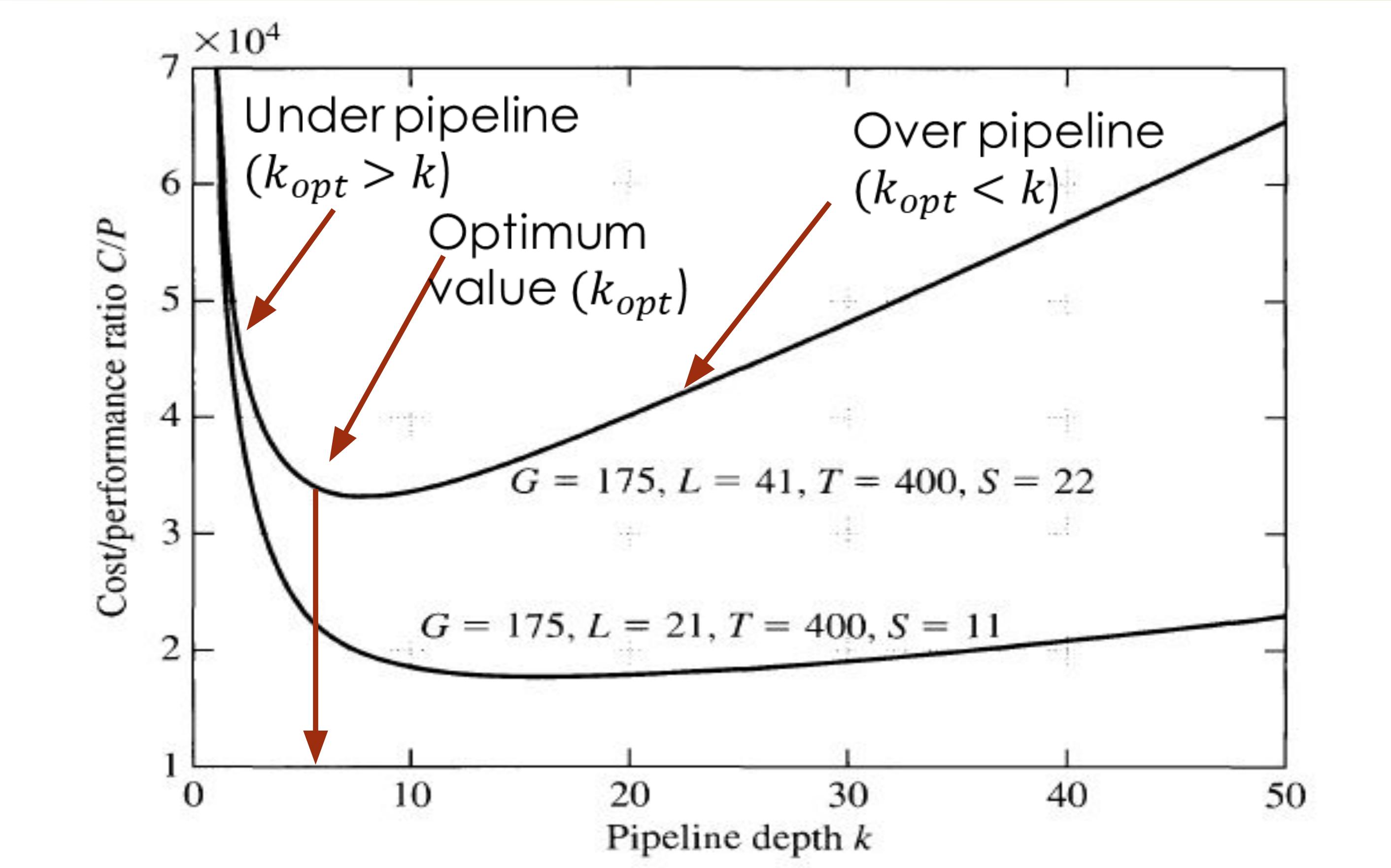
$$= LT + GS + LSk + \frac{GT}{k}$$

- Find minimum cost/performance ratio

Tradeoff between Cost and Performance

- Find minimum cost/performance ratio
- First derivative w.r.t k

$$k_{opt} = \sqrt{\frac{GT}{LS}}$$



Tradeoff between Cost and Performance

- Find minimum cost/performance ratio
- First derivative (w.r.t k)

$$k_{opt} = \sqrt{\frac{GT}{LS}}$$

- **No consideration on dynamic behavior or runtime**

Pipeline Idealism

- Motivation: k-stages pipeline increases k-fold increase in throughput
- In reality this is difficult to achieve
- Are there hidden assumptions?

Pipeline Idealism

- Are there hidden assumptions?
- Yes, 3-assumptions, called <u>pipeline idealism</u>
  - Uniform sub-computations
  - Identical computations
  - Independent computations

Pipeline Idealism: Uniform sub-computations

- The computation can be evenly partitioned into uniform-latency sub-computations
  - No (minimize) internal fragmentation
  - No (minimize) additional delay by inter-stage buffer & clocking

Pipeline Idealism: Uniform sub-computations

Example:

- Consider a module has the delay of 400-ns
- Partitioned into 3-stages with the delays
  - 125-ns, 150-ns and 125-ns
- What is the clock period?
  - 150-ns
- Inefficiency or *internal fragmentation* in the stage-1 and stage-3?
  - 25-ns

Pipeline Idealism: Uniform sub-computations

Example:

- An additional delay of 25-ns is required for proper clocking
- What is the clock period now?
  - Clock period is (150-ns plus 25-ns): 175-ns

Pipeline Idealism: Identical computations

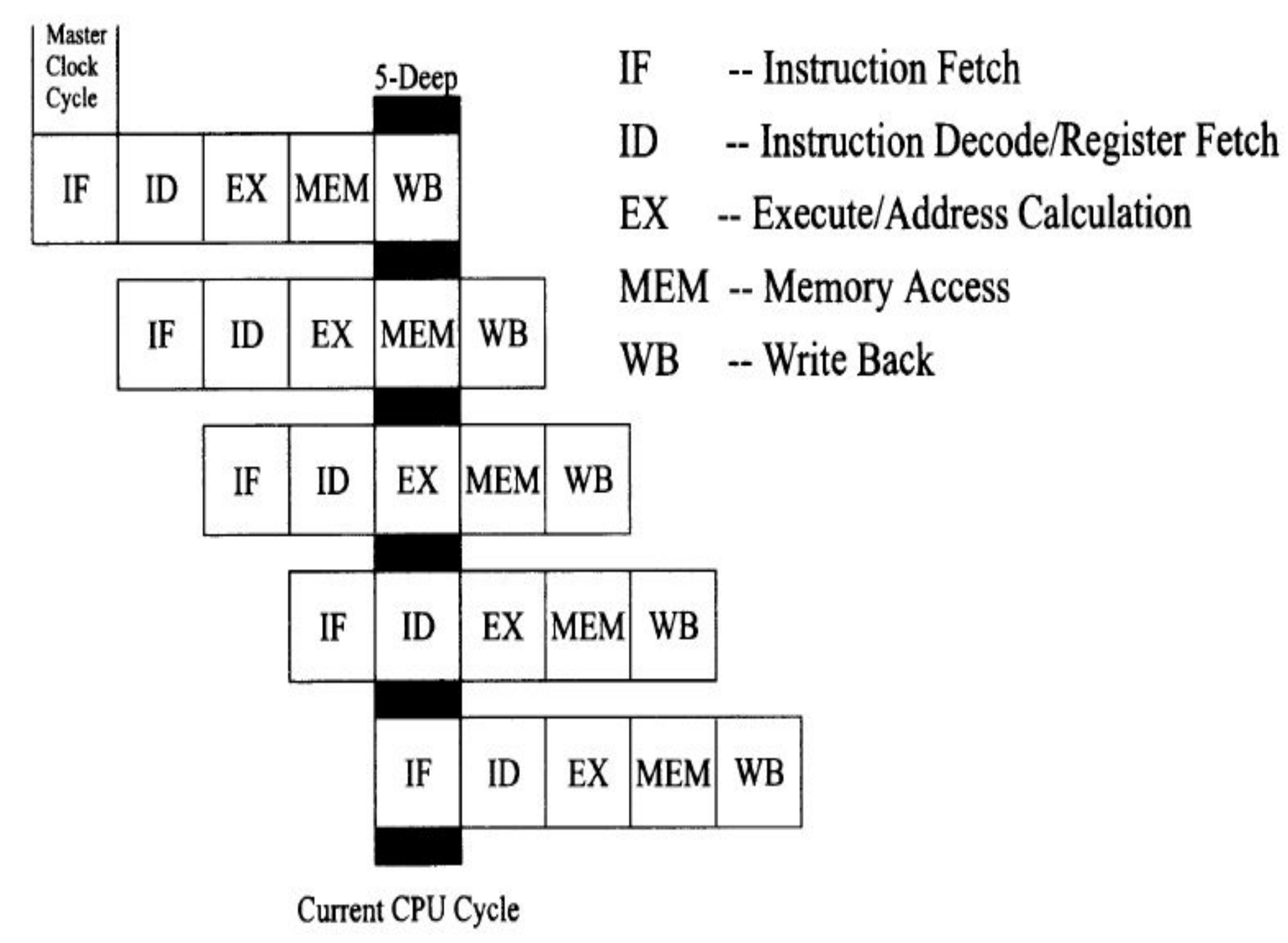
- The same computation is to be performed repeatedly for all instructions (or for all input data set)
  - Single function
  - No (minimize) external fragmentation
  - All pipeline stages are always be utilized

Pipeline Idealism: Independent Computations

- No data or control dependencies between any pair of computations
- Pipeline operates in streaming mode

Instruction pipeline or pipelined processor

- An Implementation technique
  - Exploits parallelism among the instructions
  - Overlapping the execution
- Instruction cycle
  - A logical concept
- Machine cycle
  - A physical concept
- Fill time
- Drain time



Summary

- Problems of multicycle design methodology
- Pipelined-based design methodology
- Limitations and optimum value for depth
- Pipeline idealism
- Instruction pipeline technique

Instruction Execution Strategies

- Single-cycle
  - Cycle decided by slowest instruction
- Multi-cycle
  - Unbalanced delay in the stages
- Pipelined (scaler)
  - Deeper pipelined
    - Dependencies and cache misses
- Supper-scaler
  - Dependencies and cache misses
- Out-of-order
  - Dependencies & hardwire complexity
- Super-scaler & out-of-order
  - Dependencies & hardwire complexity
- Very Large Instruction Window (VLIW)
  - Independent instructions managed by the Compiler (unaware of latencies)
- Multithreading
  - Programmer manages the parallelism
- Multi-core
  - Multiple processor (uniform/non-uniform)