Topic:

Pipelining

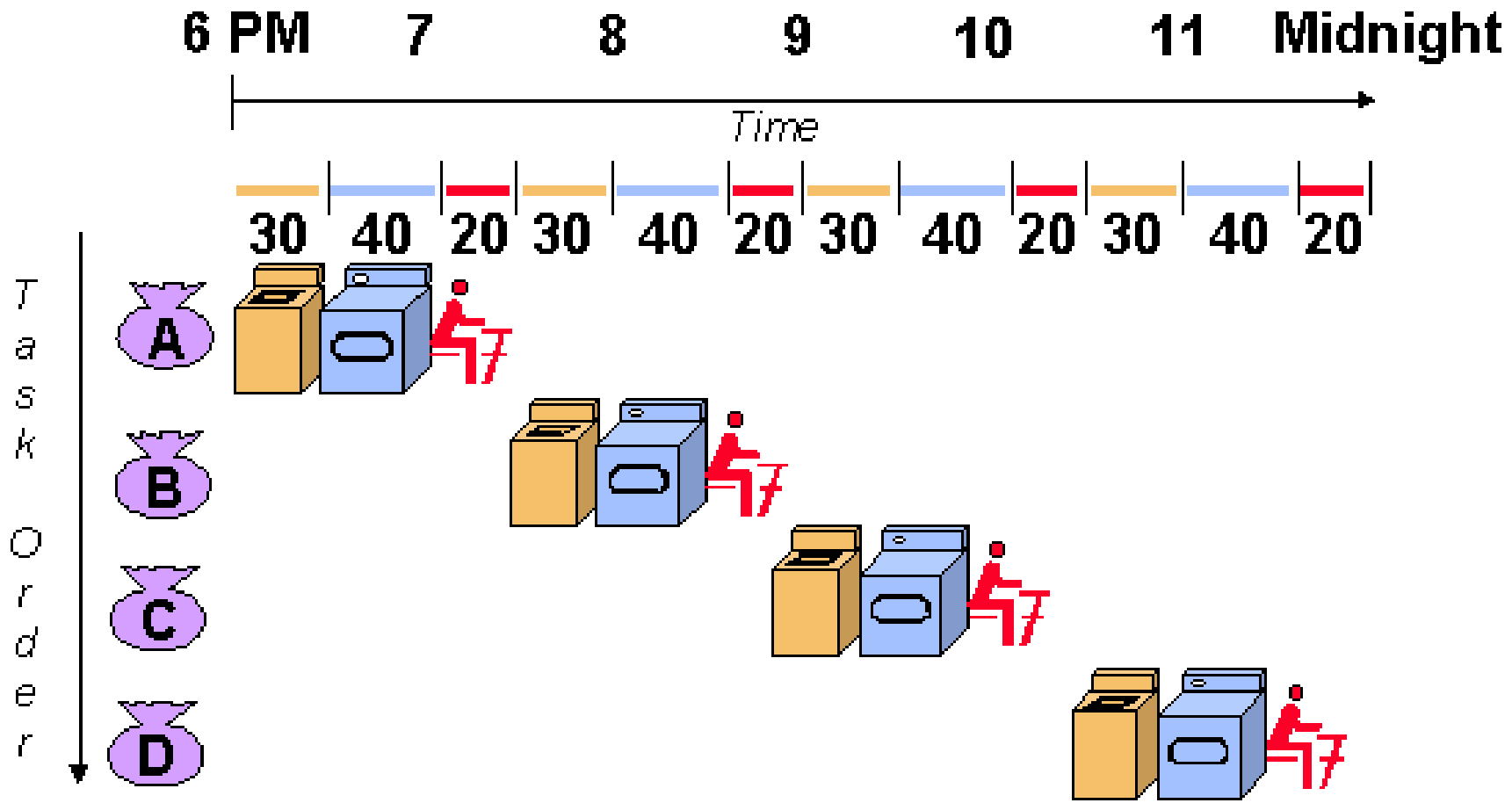
- A **Pipelining** is a series of stages, where some work is done at each stage in parallel.
- The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end.

Pipelining Case: Laundry

- 4 loads of laundry that need to washed, dried, and folded.
  - 30 minutes to wash, 40 min. to dry, and 20 min. to fold.
  - We have 1 washer, 1 dryer, and 1 folding station.

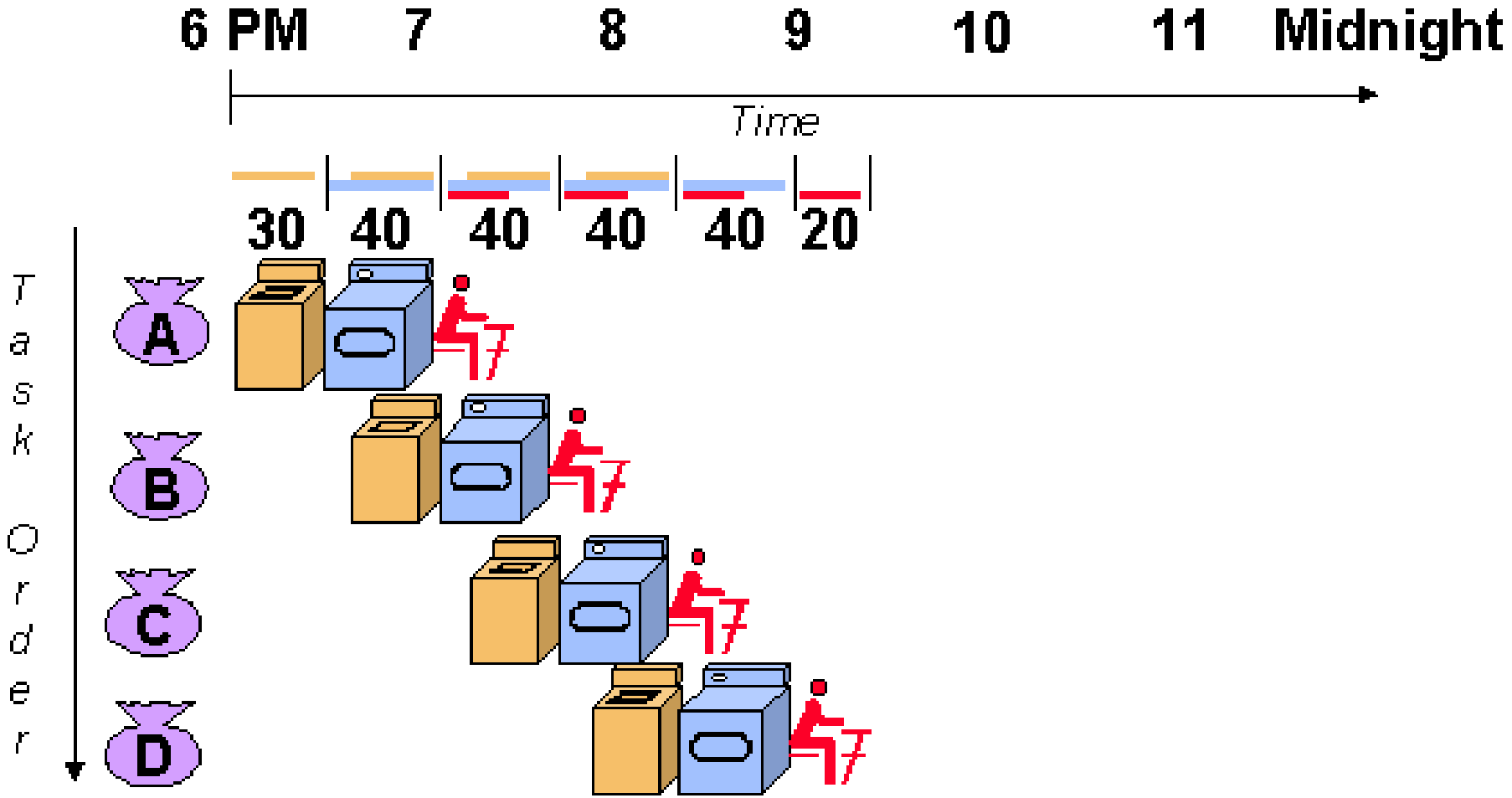
- What's the most efficient way to get the 4 loads of laundry done?

Non Pipelined Laundry



- Takes a total of 6 hours; nothing is done in parallel

Pipelined Laundry



- Using this method, the laundry would be done at 9:30.

Definition:

Pipelining is an speed up technique where multiple instructions are overlapped in execution on a processor.

Pipelining: Processors

- Computers, like laundry, typically perform the exact same
  steps for every instruction:
  - Fetch an instruction from memory
  - Decode the instruction
  - Execute the instruction
  - Read memory to get input
  - Write the result back to memory

Instruction Pipeline

- **Instruction execution process lends itself naturally to pipelining**
  - **overlap the subtasks of instruction fetch, decode and execute**

- **Instruction pipeline has six operations,**
  - Fetch instruction (FI)
  - Decode instruction (DI)
  - Calculate operands (CO)
  - Fetch operands (FO)
  - Execute instructions (EI)
  - Write result (WR)

  Overlap these operations

Instructions Fetch

- **The IF stage is responsible for obtaining the requested instruction from memory. The instruction and the program counter are stored in the register as temporary storage.**

Decode Instruction

- **The DI stage is responsible for decoding the instruction and sending out the various control lines to the other parts of the processor.**

Calculate Operands

- **The CO stage is where any calculations are performed. The main component in this stage is the ALU. The ALU is made up of arithmetic, logic and capabilities.**

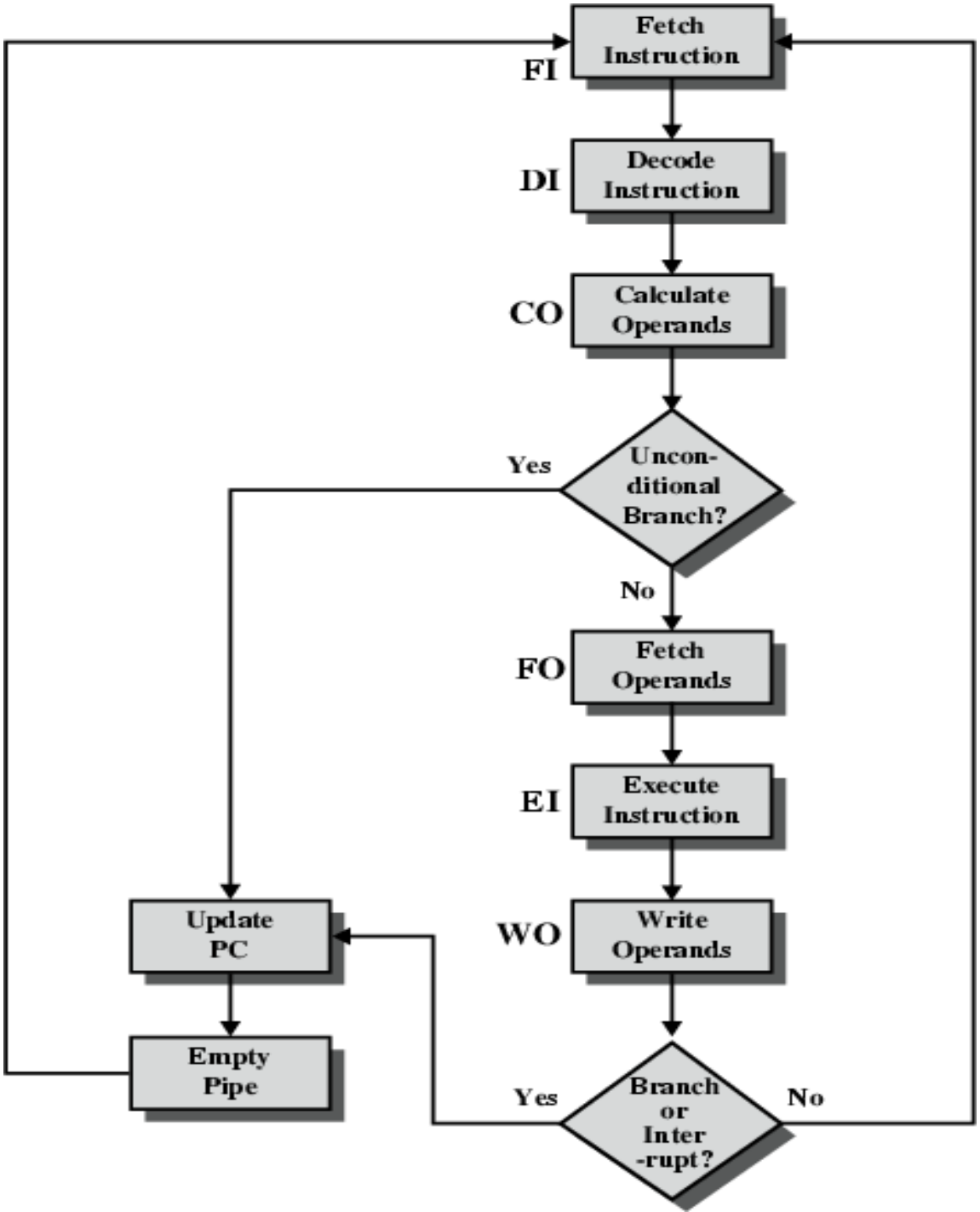
Fetch Operands and Execute Instruction

- The FO and EI stages are responsible for storing and loading values to and from memory. They also responsible for input and output from the processor respectively.

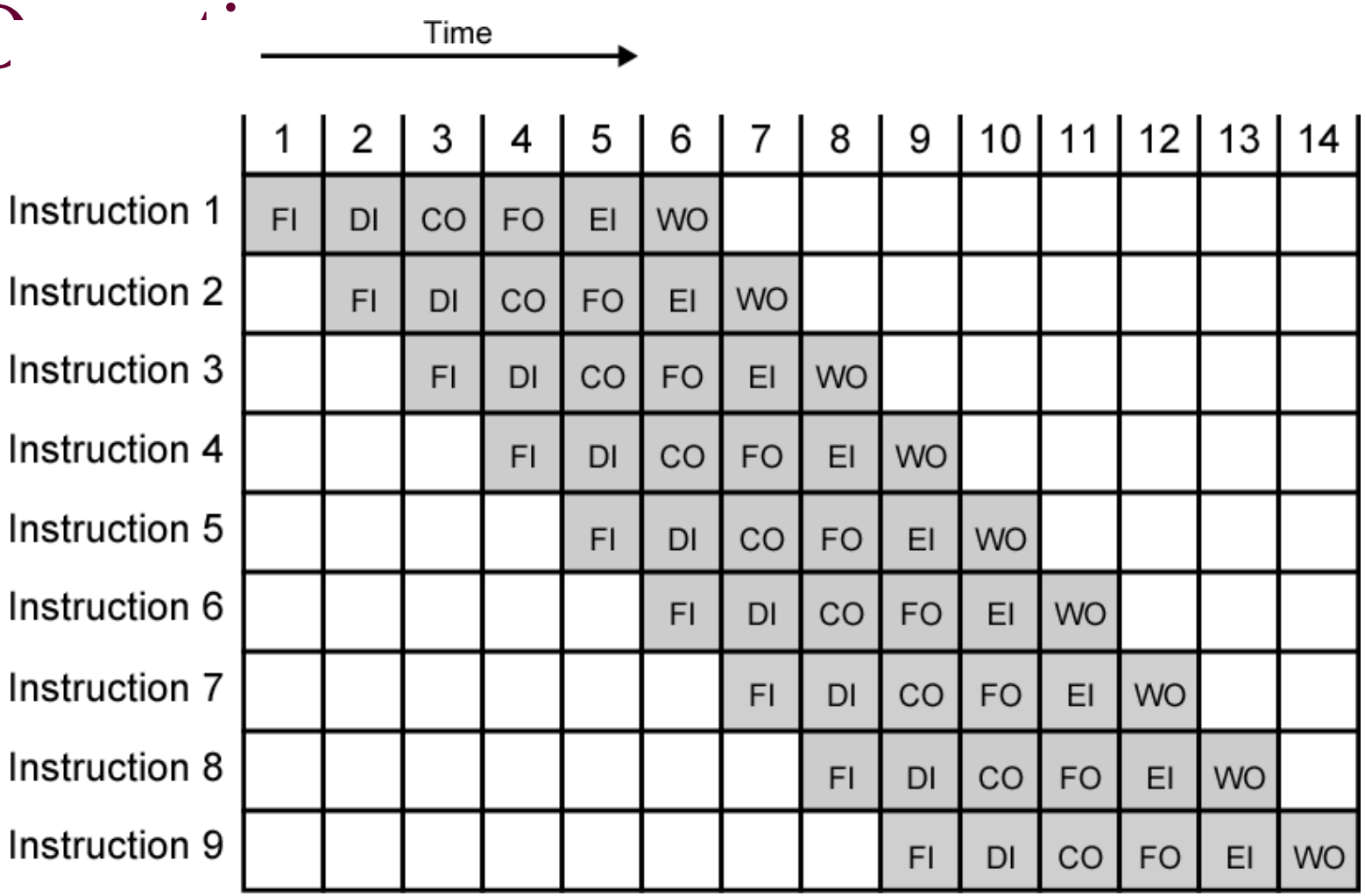
Write Operands

- The WO stage is responsible for writing the result of a calculation, memory access or input into the register file.

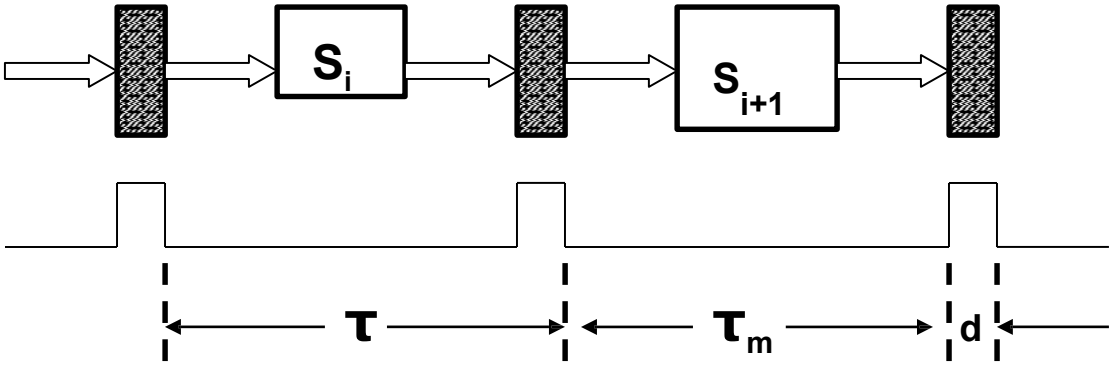
Six Stage Instruction Pipeline



Timing Diagram for Instruction Pipeline C



Pipeline Performance: Clock & Timing



Clock cycle of the pipeline : τ

Latch delay : *d*

$$\tau = \max\{\tau_m\} + d$$

Pipeline frequency : *f*

$$f = 1 / \tau$$

Pipeline Performance: Speedup & Efficiency

k-stage **pipeline processes** ***n*** **tasks in k + (n-1) clock cycles:**

k **cycles for the first task and** ***n-1*** **cycles for the remaining** ***n-1*** **tasks**

Total time to process ***n*** **tasks**

$$T_k = [\, k + (n-1)] \, \tau$$

For the non-pipelined processor

$$T_1 = n \, k \, \tau$$

Speedup factor

$$S_k = \frac{T_1}{T_k} = \frac{n \, k \, \tau}{[\, k + (n-1)] \, \tau} = \frac{n \, k}{k + (n-1)}$$

=

Advantages

- Pipelining makes efficient use of resources.
- Quicker time of execution of large number of instructions
- The parallelism is invisible to the programmer.

Can Pipelining Get Us Into Trouble?

- Yes: Pipeline Hazards
  - structural hazards: attempt to use the same resource by two different instructions at the same time
  - data hazards: attempt to use data before it is ready
    - An instruction's source operand(s) are produced by a prior instruction still in the pipeline
  - control hazards: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - branch instructions

- Can always resolve hazards by waiting
  - pipeline control must detect the hazard
  - and take action to resolve hazards