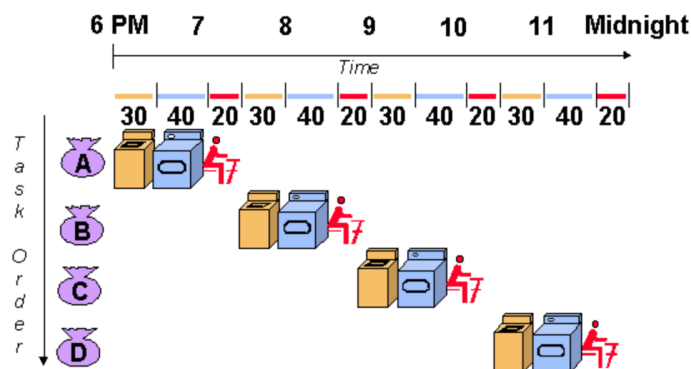


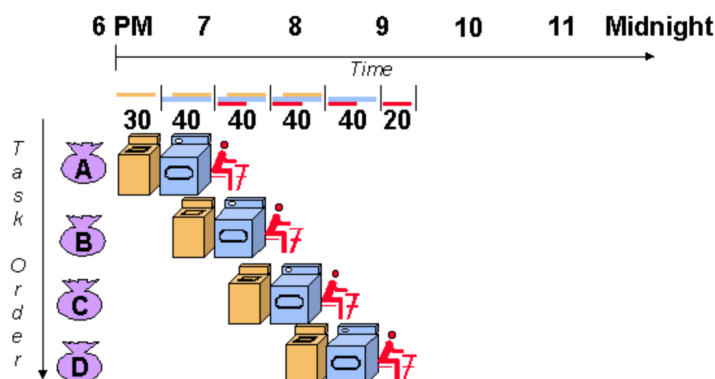
Pipelining (I)

What is pipelining?

- We mentioned pipelining when discussing RISC machines. We say that RISC can leverage the pipelining to improve its instruction execution speed.
- So, what is pipelining?
 - A technique for **implementing instruction-level parallelism within a single processor**.
 - It tries to keep every part of the processor busy by dividing incoming instructions into a series of sequential steps performed by different processor units with different parts of the instructions processed in parallel.
- The idea may be too abstract to understand. Let's use a real life example to illustrate it.
- Assume I have four loads of dirty laundry that need to be washed, dried, and folded. There is only one washer, one dryer, and I'm the only one who can fold. The washer takes 30 minutes to wash a load, dryer takes 40 minutes to dry a load, and it takes me 20 minutes to fold a load. How long it will take to do the four loads of dirty laundry?
 - *Method one*: wash, dry, and fold each laundry load sequentially
 - Slow: 6 hours for 4 loads $(30 + 40 + 20) * 4$



- *Method two*: wash, dry, and fold perform in parallel
 - Fast: 3.5 hours for 4 loads $30 + 40 * 4 + 20$

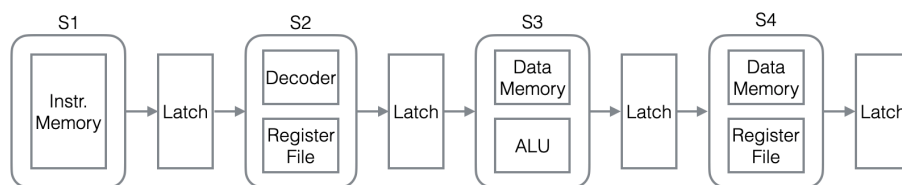


- So, if we see the washer, dryer, and me folding as analogous to different part of the processor and see the dirty laundry loads as analogous to instructions, we then can implement pipelining by letting multiple instructions operate simultaneously using different resources.
- Note that **pipelining doesn't improve the latency of a single instruction**, but it **improves the throughput of entire workload**.
 - Latency refers to the amount of time between when the instruction is issued and when it is completed.
 - Throughput refers to the number of instructions that complete in a certain amount of time.
- **Pipelining performance is limited by the slowest step**. E.g., dryer in the above example.
- **Unbalanced lengths of pipelining steps reduce the performance**.

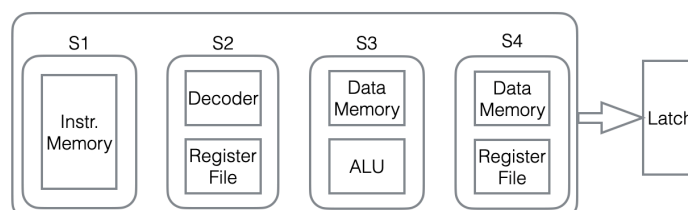
Multiple-stage vs Single-stage CPU

- Pipelining allows instructions be executed in parallel, so that programs can be processed more quickly.
- To process an instruction, a RISC CPU has several steps. E.g. fetch, decode, execute, and write back.
- Single-stage CPU: all steps are one stage
- Multiple-stage CPU: steps are in multiple stages
 - One latch layer for each stage to save temporary instruction result from the previous stage.

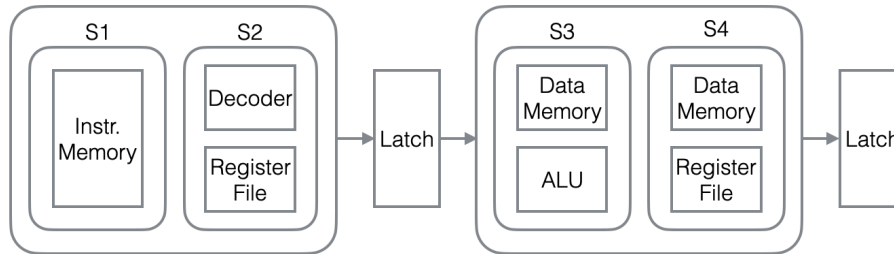
Multiple-stage CPU



Single-stage CPU



- Assume that time spent on each step (S1 ~ S4) is the same, which means if S1 takes 5 ns in multiple-stage, then it also takes 5 ns in single-stage. Assume all the latches take the same amount of time as well.
- What may be the layout if it is a 2-stage CPU? Notes: the balanced length of each stage.



- It seems like that the multiple-stage CPU takes more time than the single-stage CPU due to the extra latch time. Why we want to use the multiple-stage CPU?
- The single-stage CPU is like the all-in-one laundry combo unit. You have to do the dirty laundry loads one by one (sequentially). While the multiple-stage CPU is like the separate washer and dryer. You can do the dirty laundry loads in parallel. Although it takes a few minutes to put the washed load into the dryer, the time is less than the time it takes to wash or dry a load.