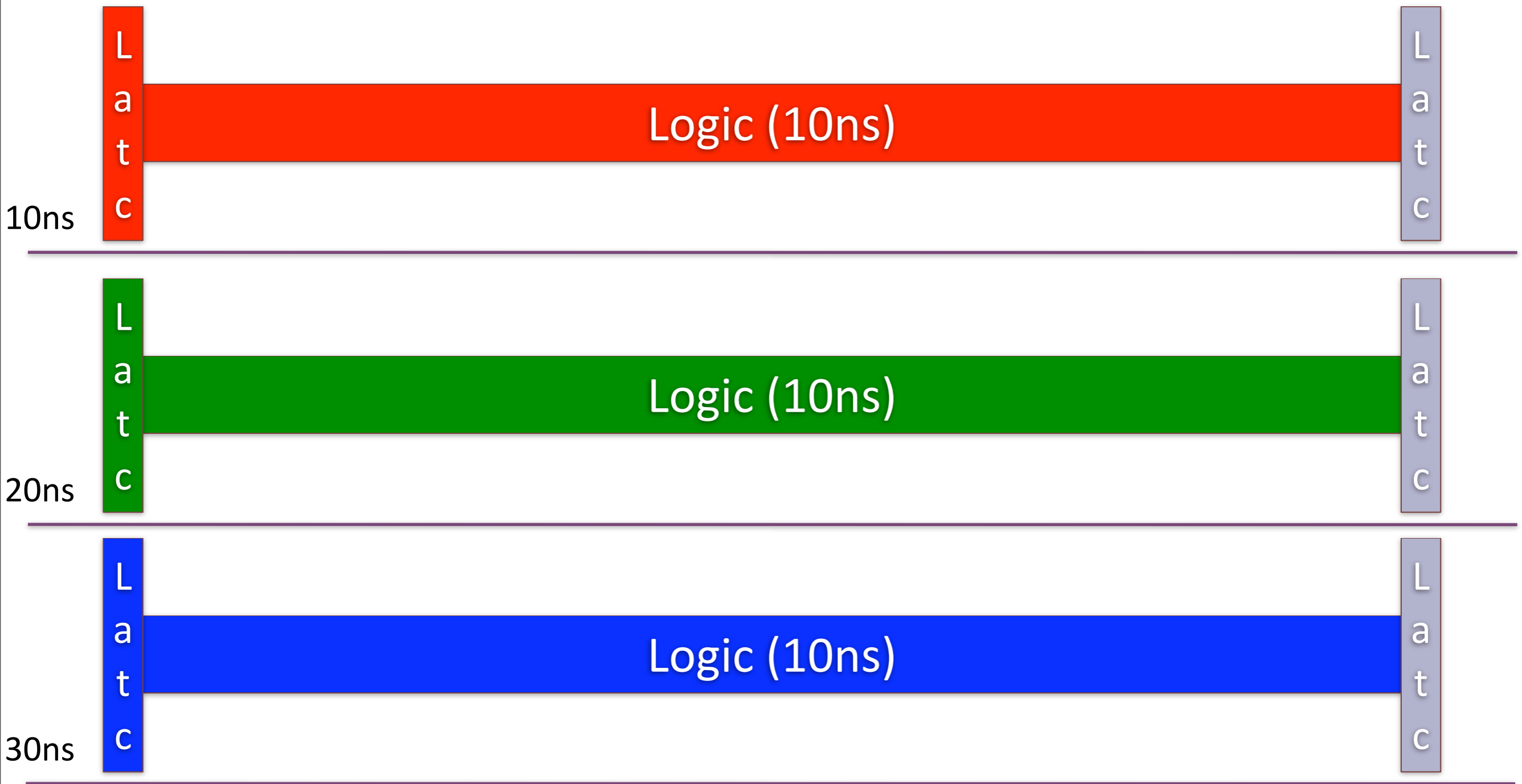


Pipelining

Today

- Quiz
- Introduction to pipelining

Pipelining

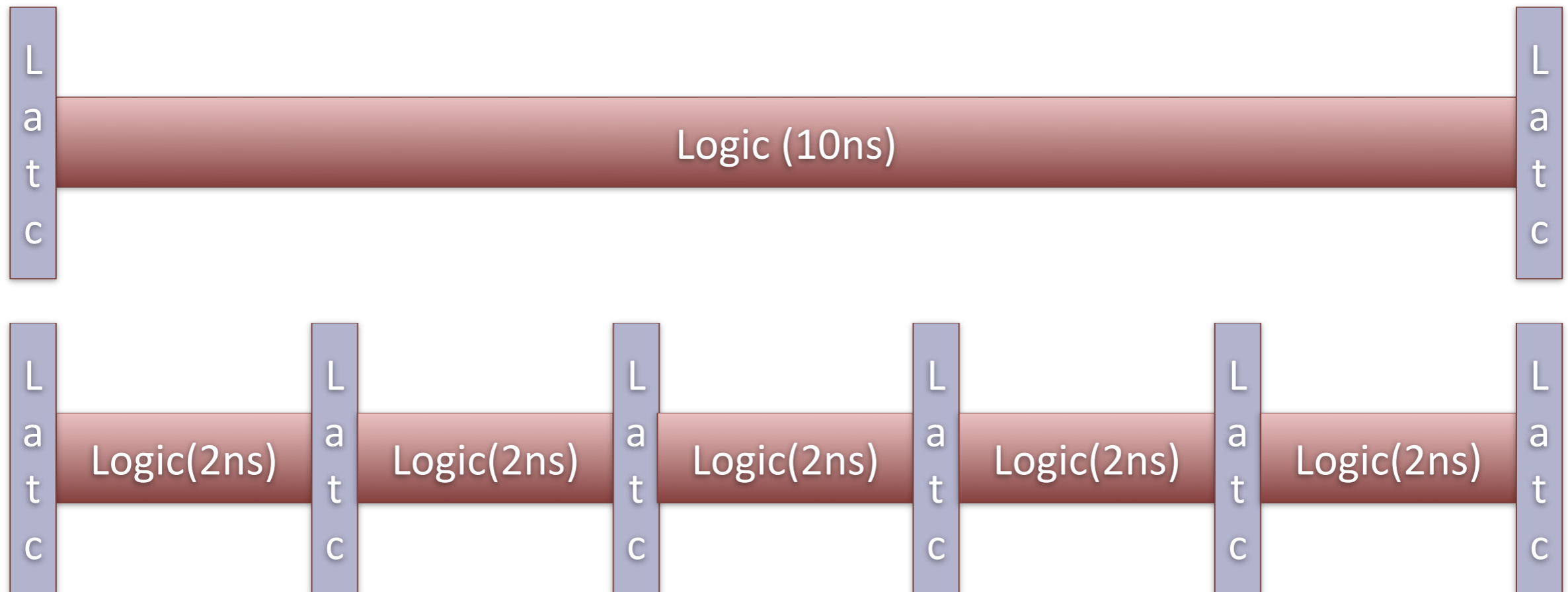


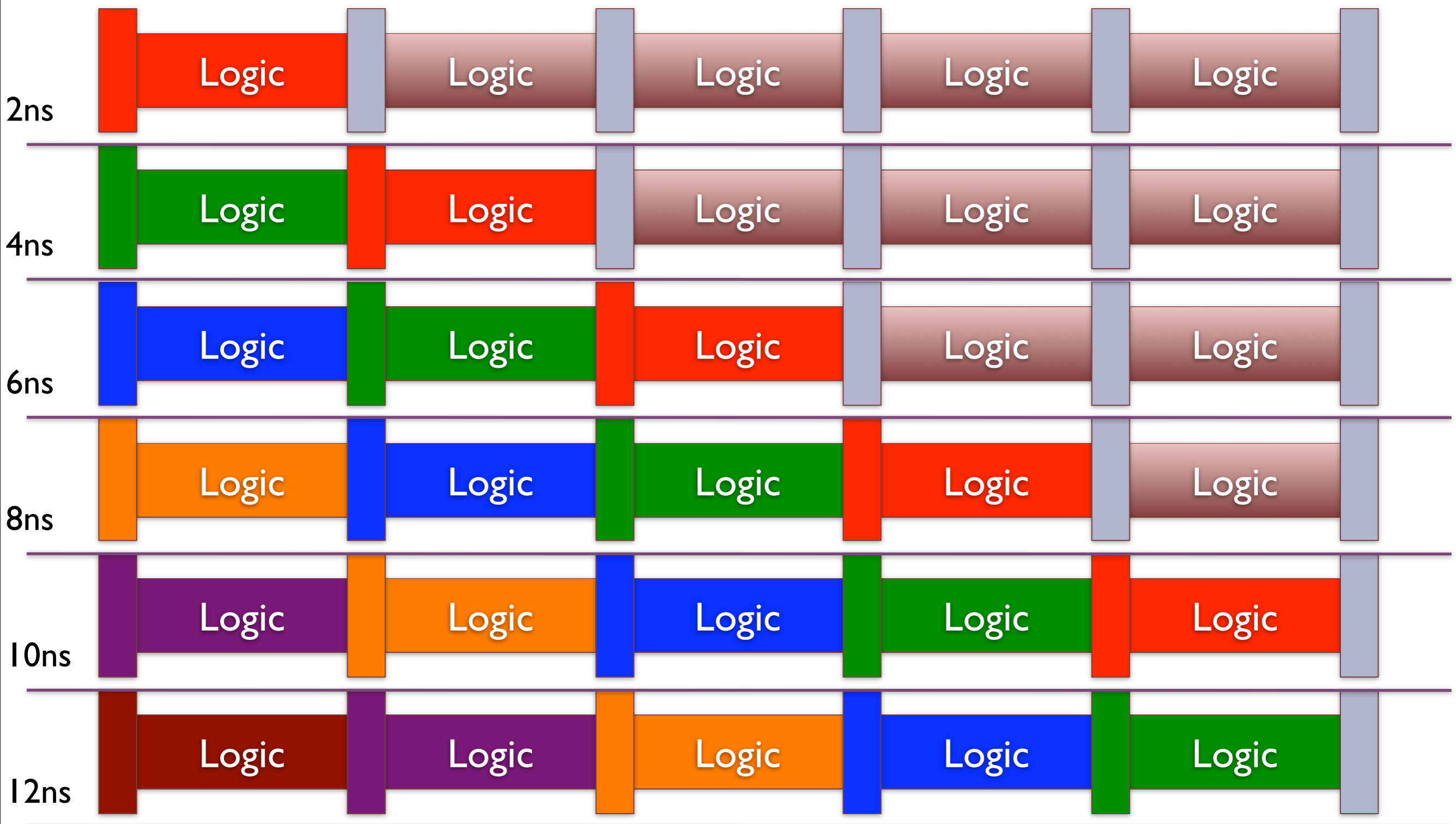
What's the latency for one unit of work?

What's the throughput?

Pipelining

1. Break up the logic with latches into "pipeline stages"
2. Each stage can act on different data
3. Latches hold the inputs to their stage
4. Every clock cycle data transfers from one pipe stage to the next



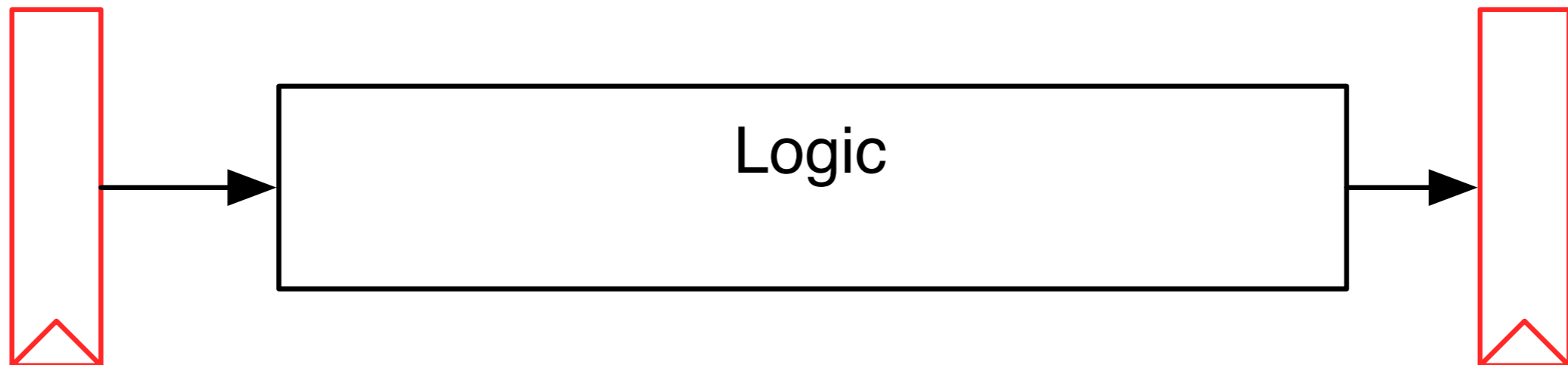


What's the latency for one unit of work?

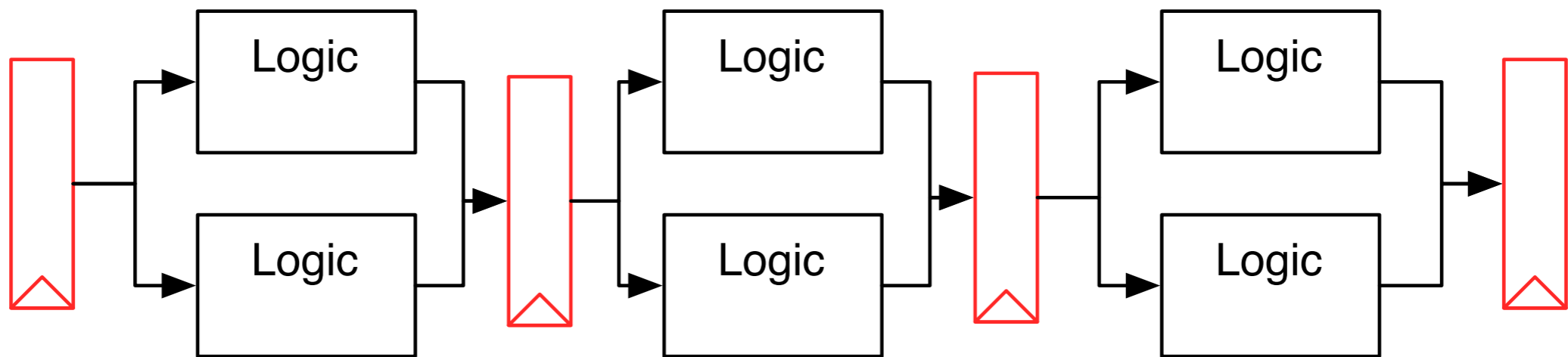
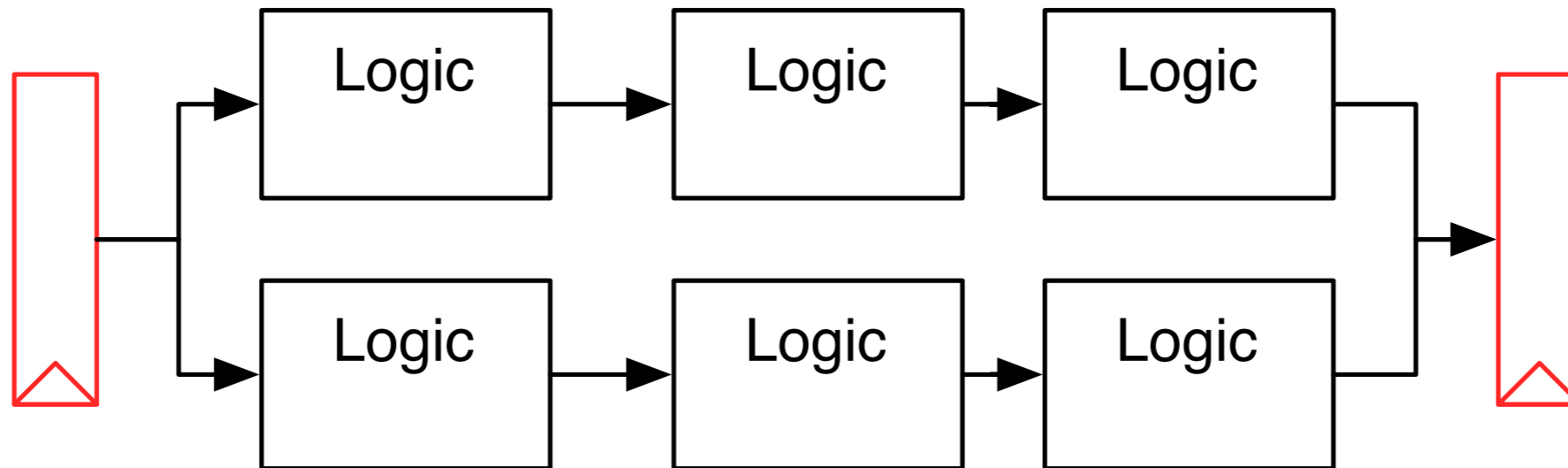
What's the throughput?

Critical path review

- Critical path is the longest possible delay between two registers in a design.
- The critical path sets the cycle time, since the cycle time must be long enough for a signal to traverse the critical path.
- Lengthening or shortening non-critical paths does not change performance
- Ideally, all paths are about the same length

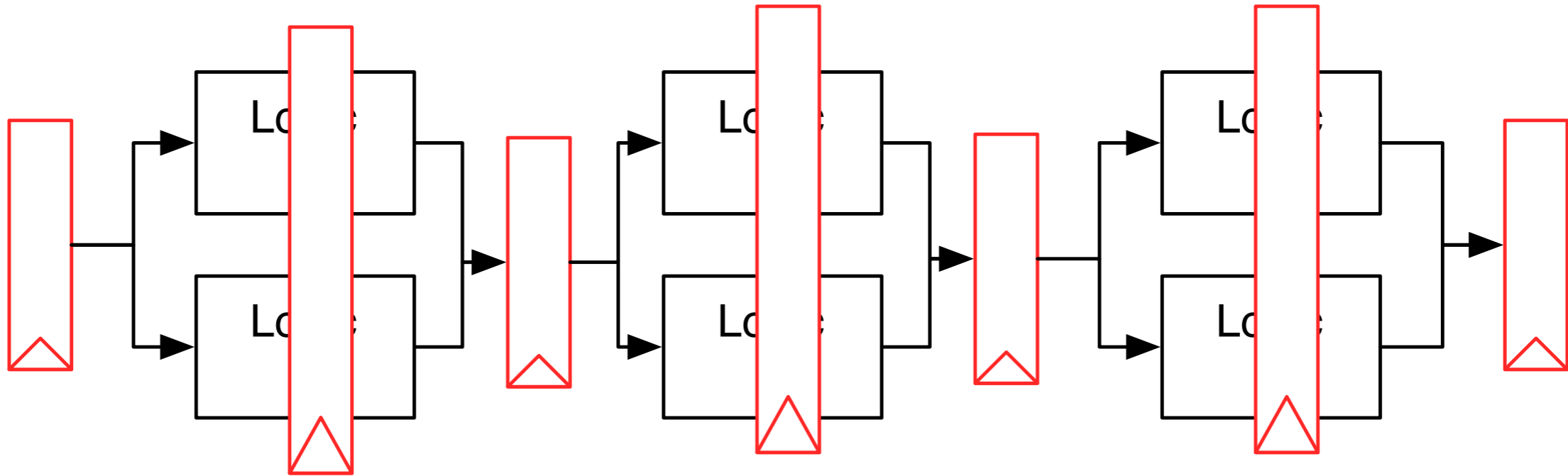


Pipelining and Logic



- Hopefully, critical path reduced by 1/3

Limitations of Pipelining



- You cannot pipeline forever
 - Some logic cannot be pipelined arbitrarily -- Memories
 - Some logic is inconvenient to pipeline.
 - How do you insert a register in the middle of an adder?
- Registers have a cost
 - They cost area -- choose “narrow points” in the logic
 - They cost time
 - Extra logic delay
 - Set-up and hold times.
- Pipelining may not affect the critical path as you expect
 - It is difficult to balance path lengths.

Pipelining Overhead

- Logic Delay (LD) -- How long does the logic take (i.e., the useful part)
- Set up time (ST) -- How long before the clock edge do the inputs to a register need be ready?
- Register delay (RD) -- Delay through the internals of the register.

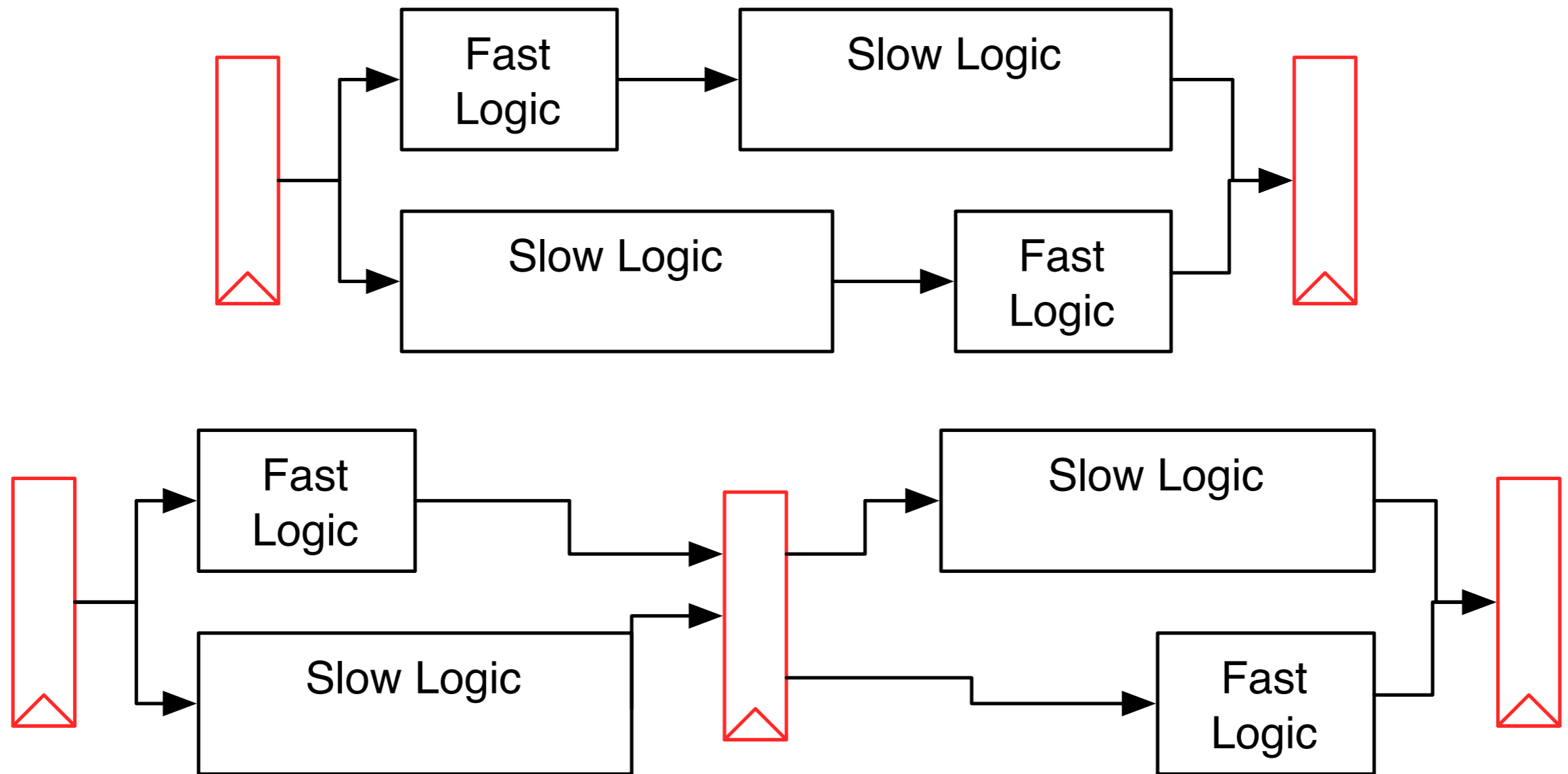
- BaseCT -- cycle time before pipelining
 - $\text{BaseCT} = \text{LD} + \text{ST} + \text{RD}$.
 - Total delay = BaseCT
- PipeCT -- cycle time after pipelining N times
 - PipeCT = ?
 - Total delay = ?

Pipelining Overhead

- Logic Delay (LD) -- How long does the logic take (i.e., the useful part)
- Set up time (ST) -- How long before the clock edge do the inputs to a register need be ready?
- Register delay (RD) -- Delay through the internals of the register.

- BaseCT -- cycle time before pipelining
 - $\text{BaseCT} = \text{LD} + \text{ST} + \text{RD}$.
- PipeCT -- cycle time after pipelining N times
 - $\text{PipeCT} = \text{ST} + \text{RD} + \text{LD}/N$
 - $\text{Total time} = N * \text{ST} + N * \text{RD} + \text{LD}$

Pipelining Difficulties

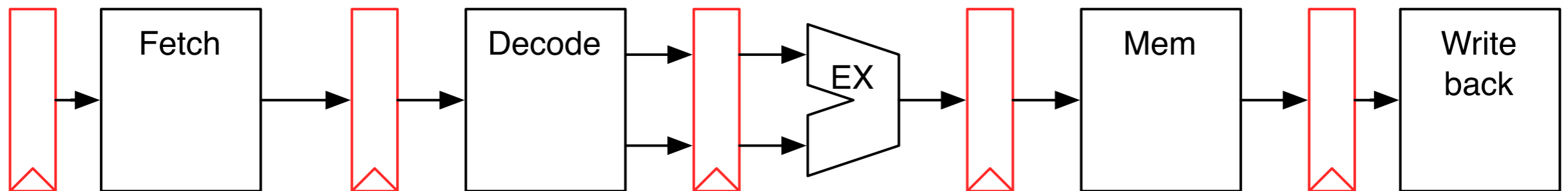
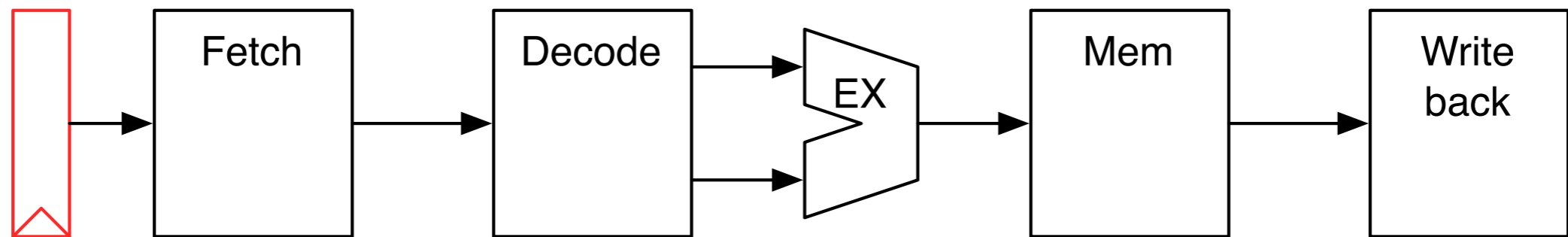


- The critical path only went down “fast logic”

How to pipeline a processor

- Break each instruction into pieces -- remember the basic algorithm for execution
 - Fetch
 - Decode
 - Collect arguments
 - Execute
 - Write back results
 - Compute next PC
- The “classic 5-stage MIPS pipeline”
 - Fetch -- read the instruction
 - Decode -- decode and read from the register file
 - Execute -- Perform arithmetic ops and address calculations
 - Memory -- access data memory.
 - Write back-- Store results in the register file.

Pipelining a processor



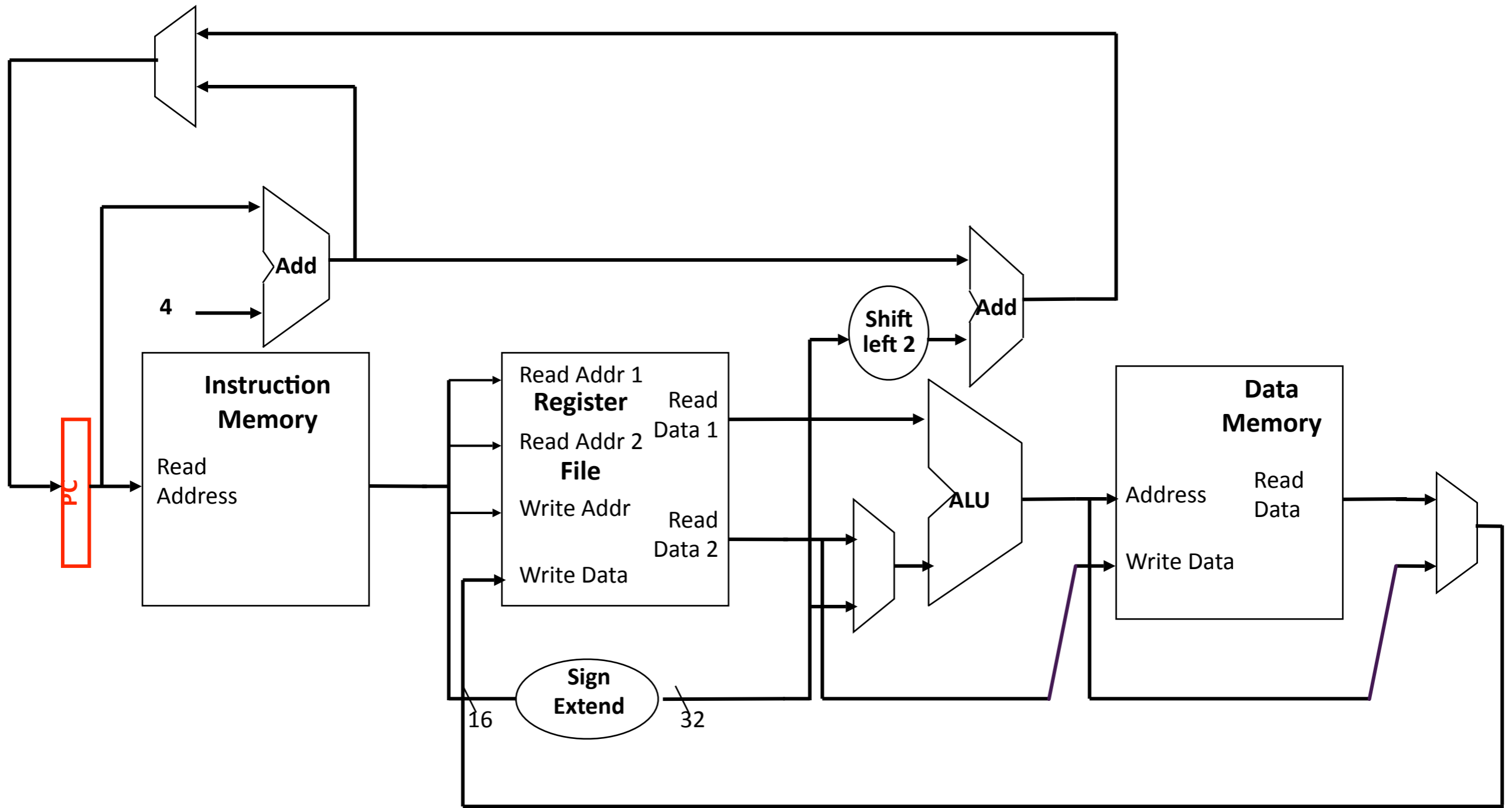
Impact of Pipelining

- Break the processor into P pipe stages
- What happens to latency?
 - $L = \text{Inst} * \text{CPI} * \text{CycleTime}$
- The cycle time = ?
- CPI = ?

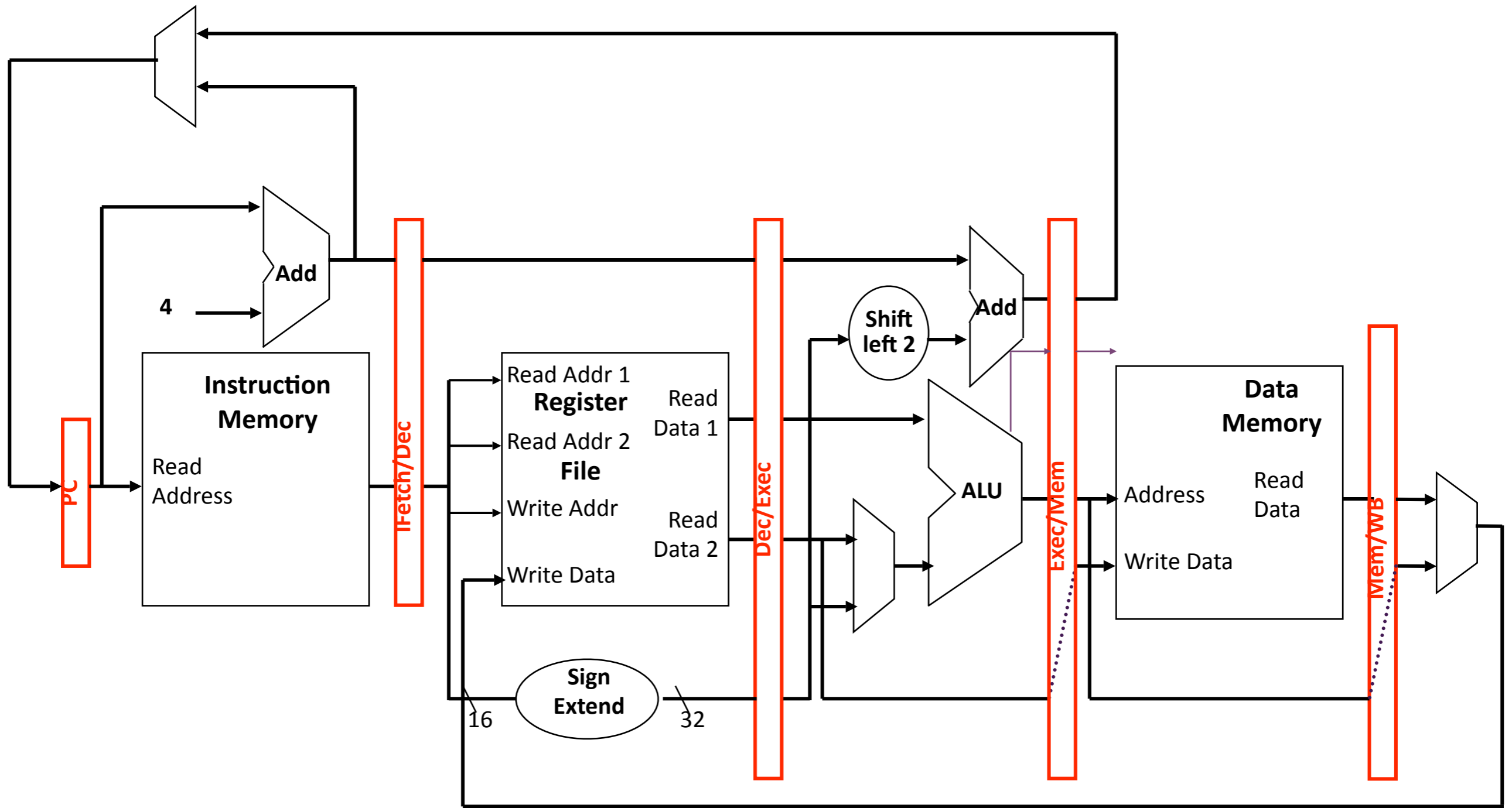
Impact of Pipelining

- Break the processor into P pipe stages
- What happens to latency?
 - $L = \text{Inst} * \text{CPI} * \text{CycleTime}$
- The cycle time = CT/P
- $\text{CPI} = 1$
 - CPI is an average: Cycles/instructions
 - When # of instructions is large, $\text{CPI} = 1$
 - If just one instruction, $\text{CPI} = P$

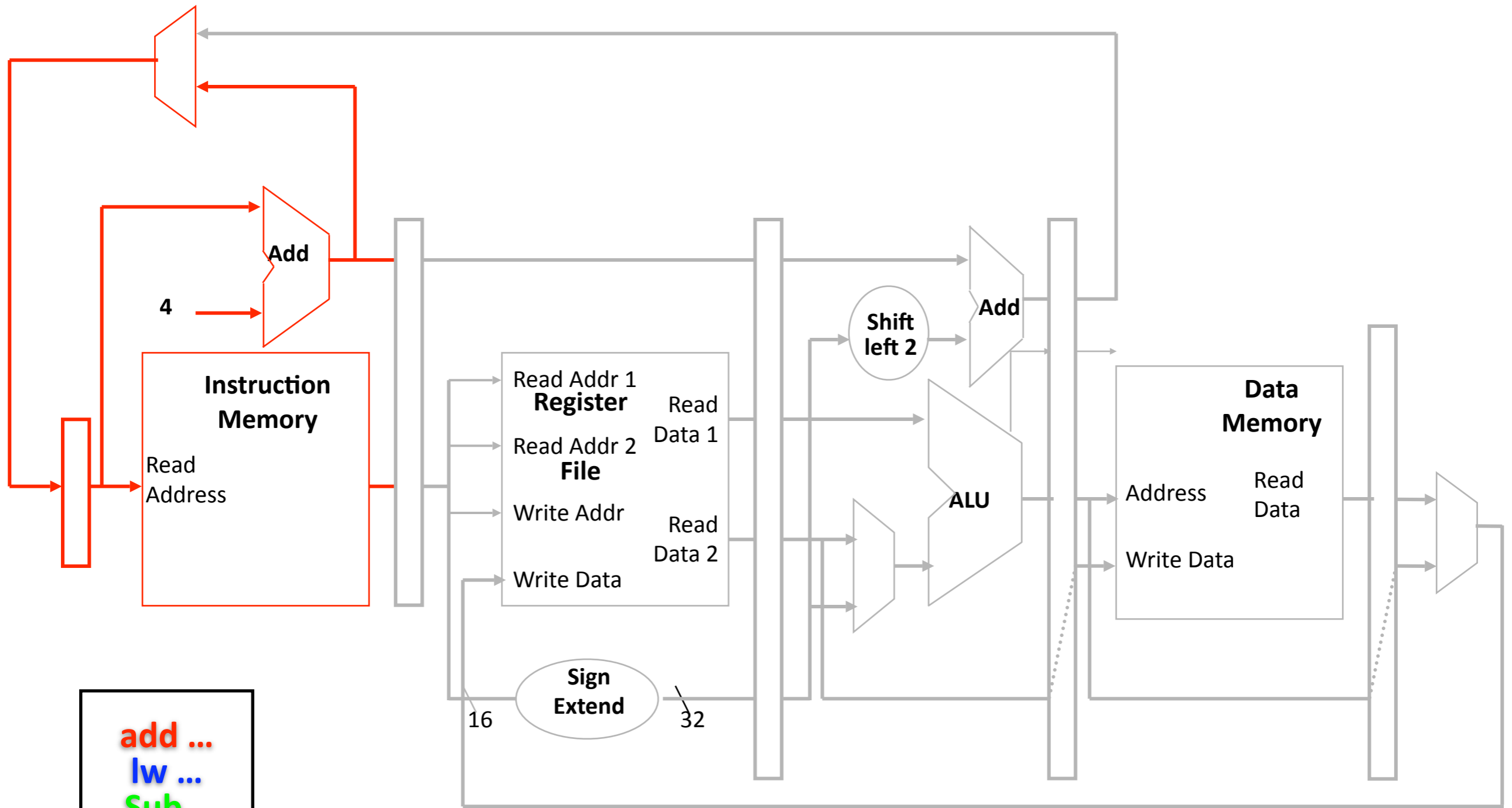
Pipelined Datapath



Pipelined Datapath

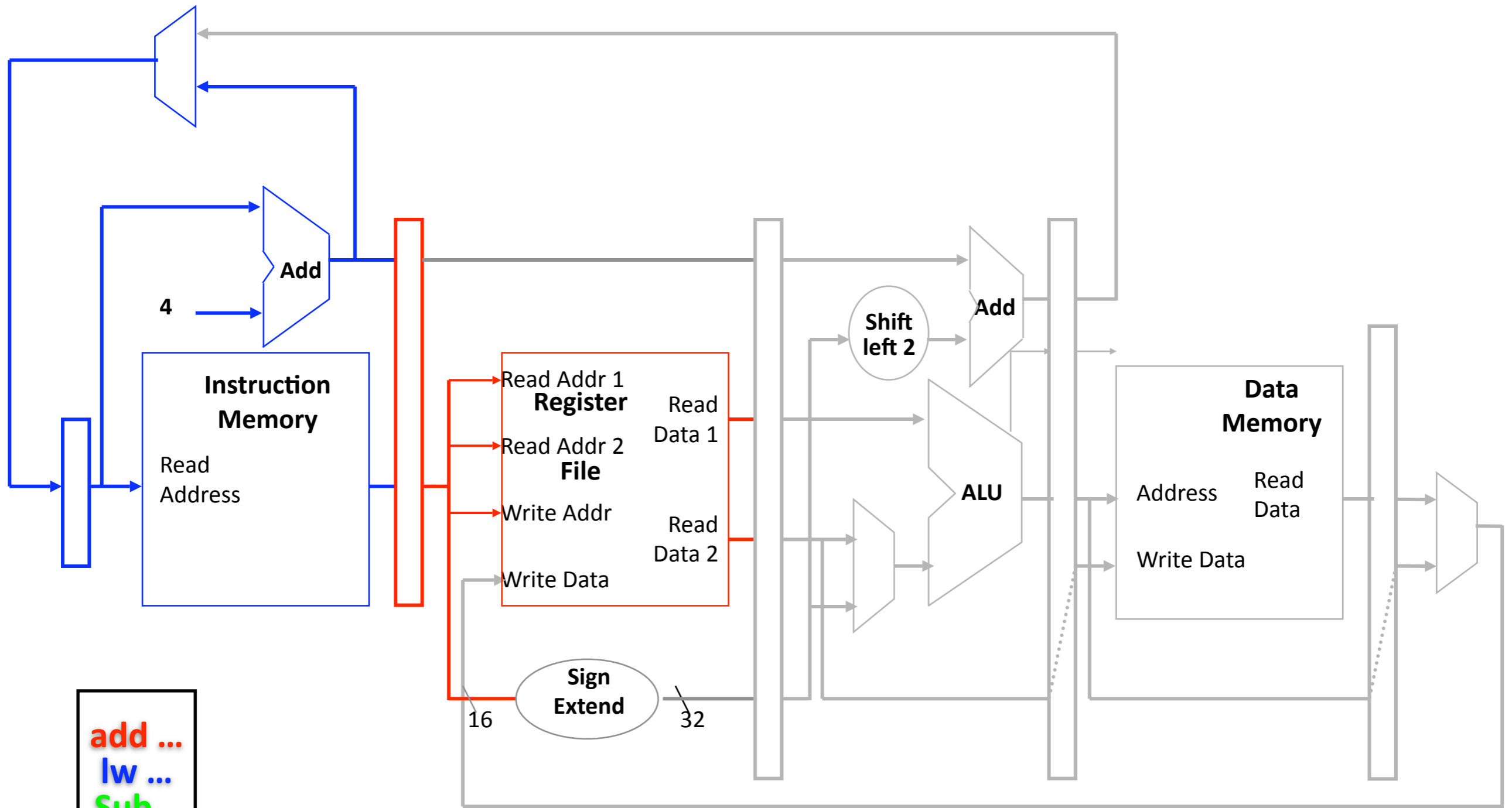


Pipelined Datapath

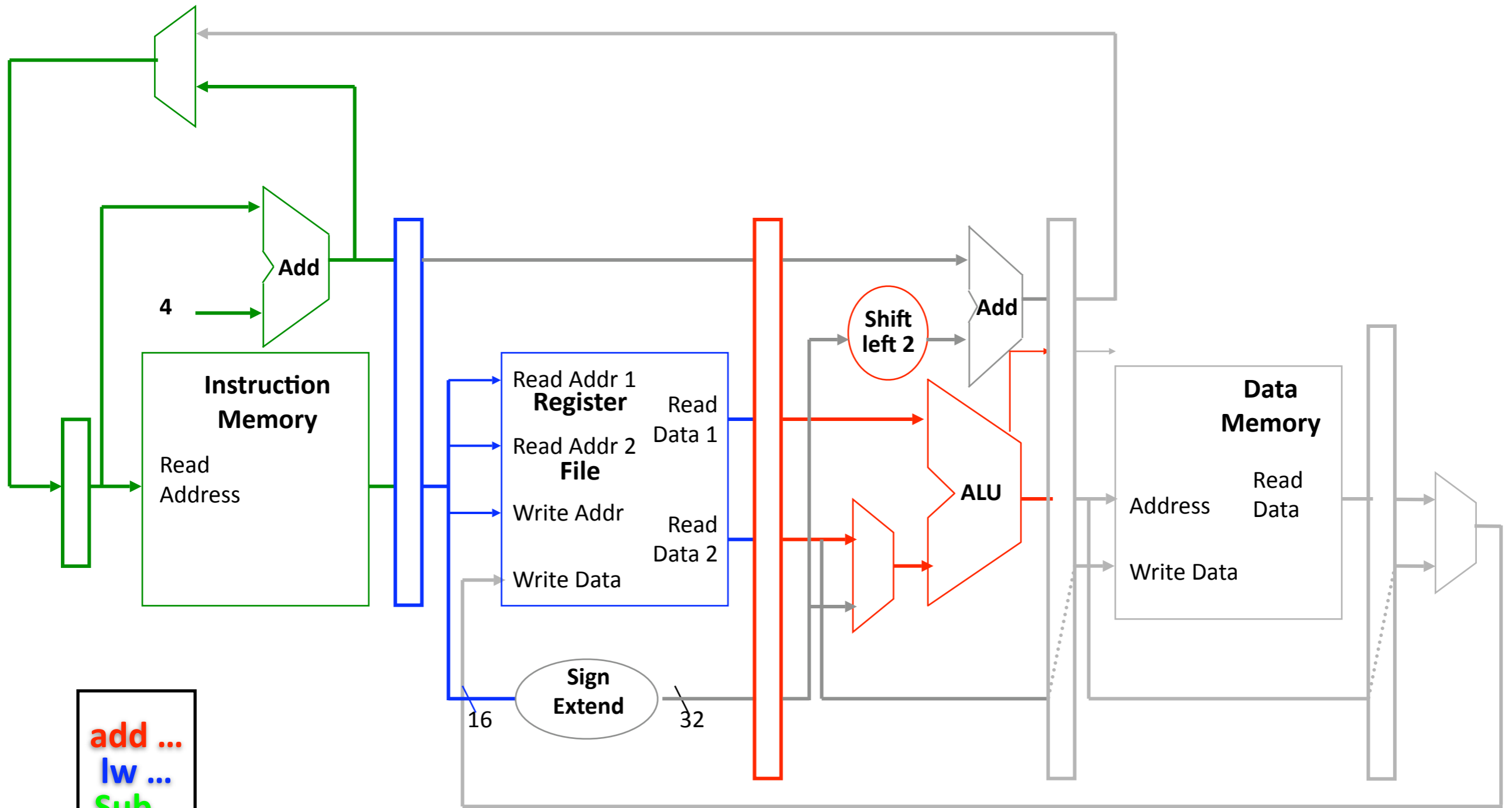


- add ...
- lw ...
- Sub...
- Sub
- Add ...
- Add ...

Pipelined Datapath

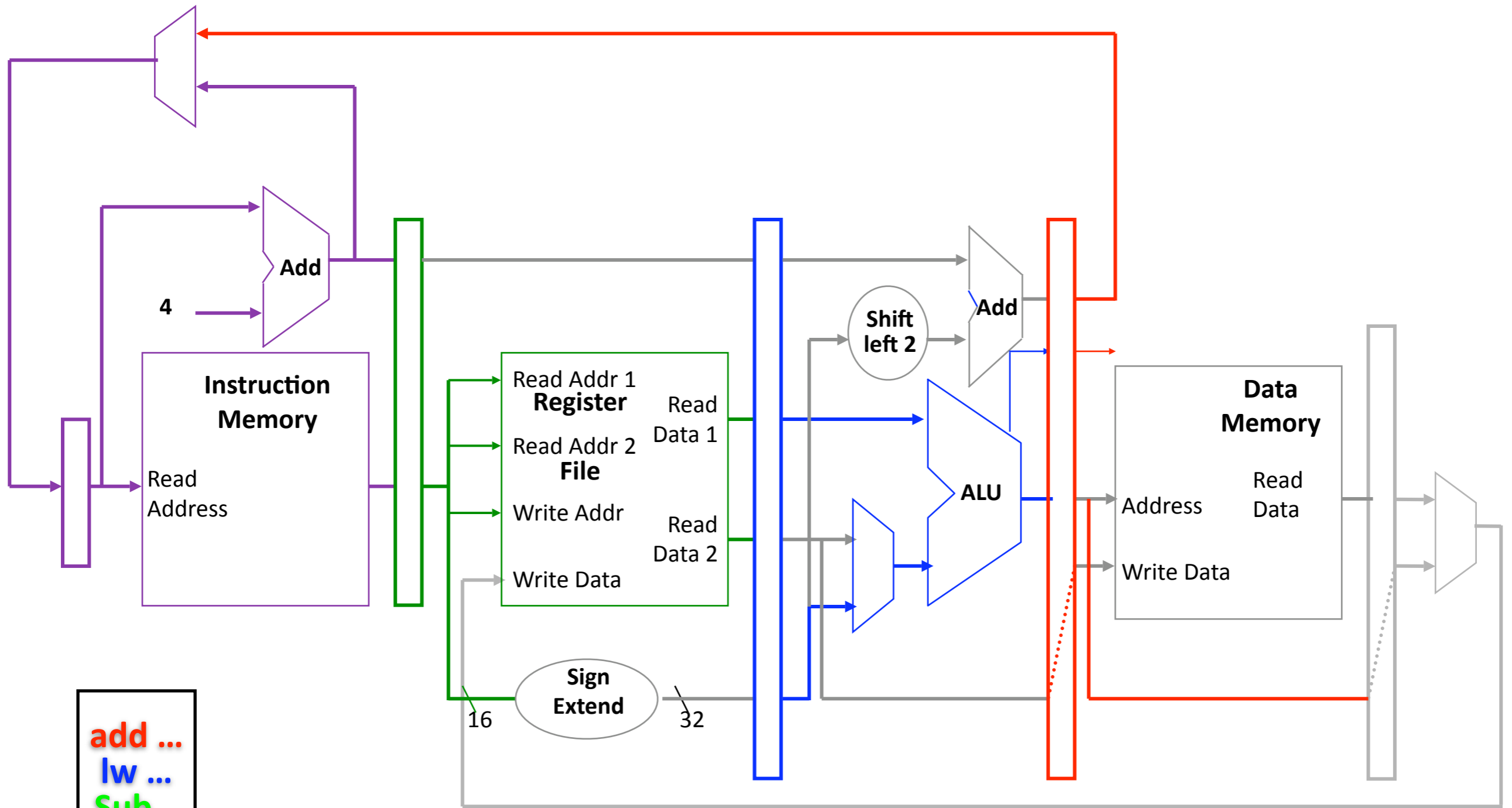


Pipelined Datapath



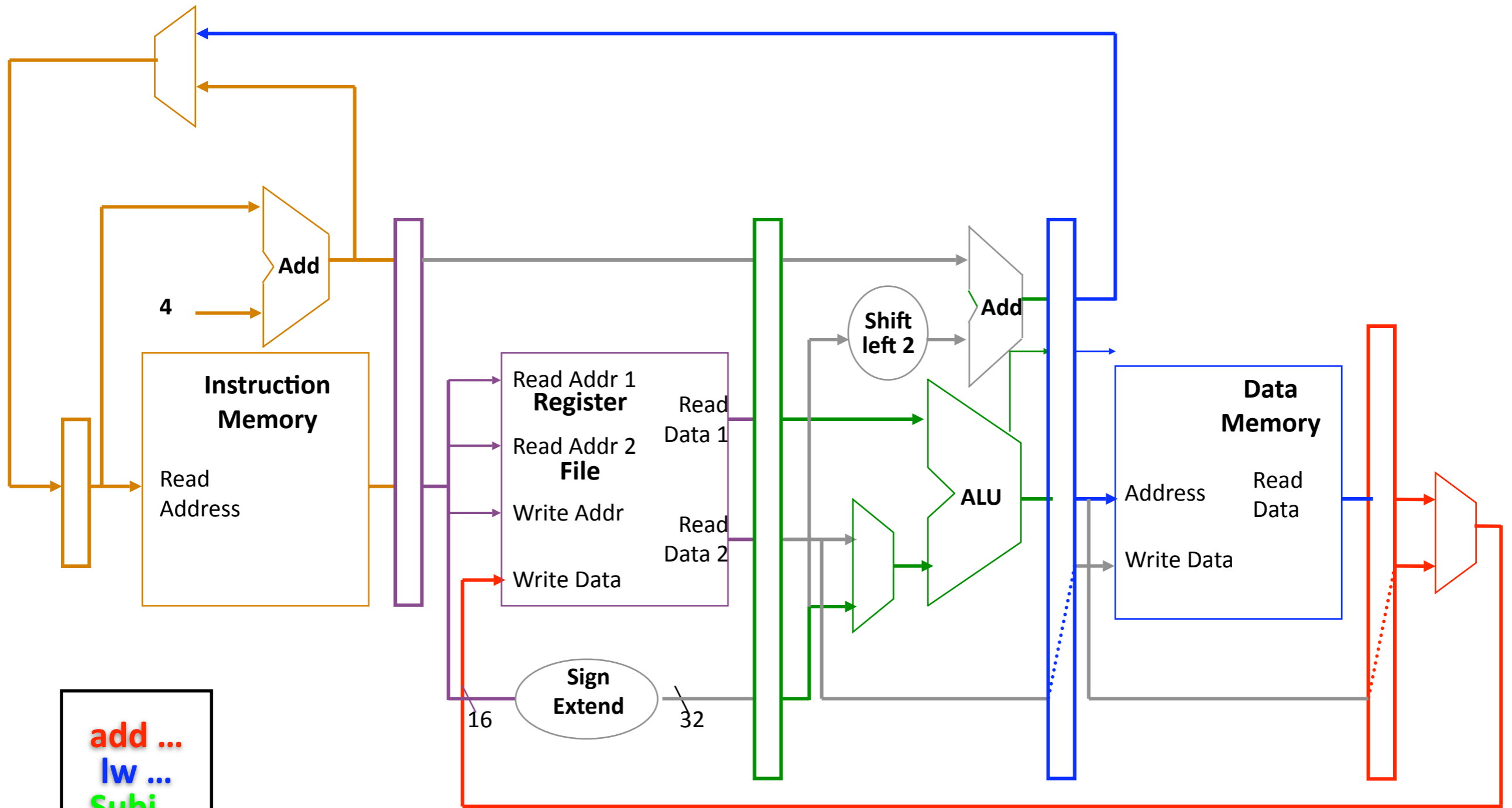
- add ...
- lw ...
- Sub...
- Sub
- Add ...
- Add ...

Pipelined Datapath



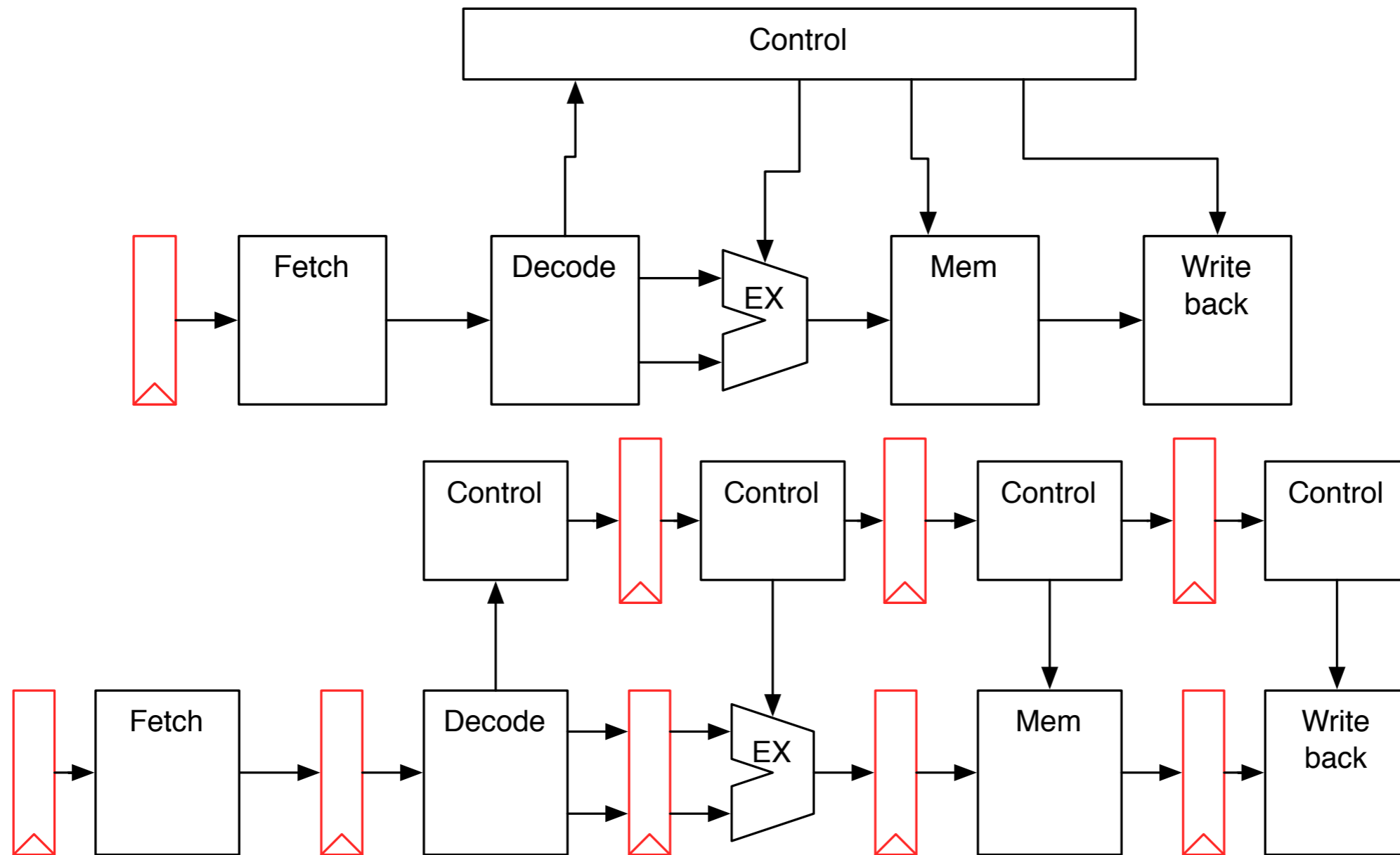
- add ...
- lw ...
- Sub...
- Sub
- Add ...
- Add ...

Pipelined Datapath



- add ...
- lw ...
- Subi...
- Sub
- Add ...
- Add ...

Simple Pipelining Control



- Compute all the control bits in decode, then pass them from stage to stage. It won't stay this simple...

Pipelining is Tricky

- If all the data flows in one direction, pipelining is relatively easy.
- Not so, for processors.
 - Decode and write back both access the register file.
 - Branch instructions affect the next PC
 - Instructions need values computed by previous instructions

Not just tricky, Hazardous!

- Hazards are situations where pipelining does not work as elegantly as we would like
 - Caused by backward flowing signals
 - Or by lack of available hardware
- Three kinds
 - Data hazards -- an input is not available on the cycle it is needed
 - Control hazards -- the next instruction is not known
 - Structural hazards -- we have run out of a hardware resource
- Detecting, avoiding, and recovering from these hazards is what makes processor design hard.
 - That, and the Xilinx tools ;-)

A Structural Hazard

- Both the decode and write back stage have to access the register file.
- There is only one registers file. A structural hazard!!
- Solution: Write early, read late
 - Writes occur at the clock edge and complete long before the end of the cycle
 - This leave enough time for the outputs to settle for the reads.
- Hazard avoided!

