Lecture 7 slides

CE/CZ 3001:
Advanced Computer Architecture

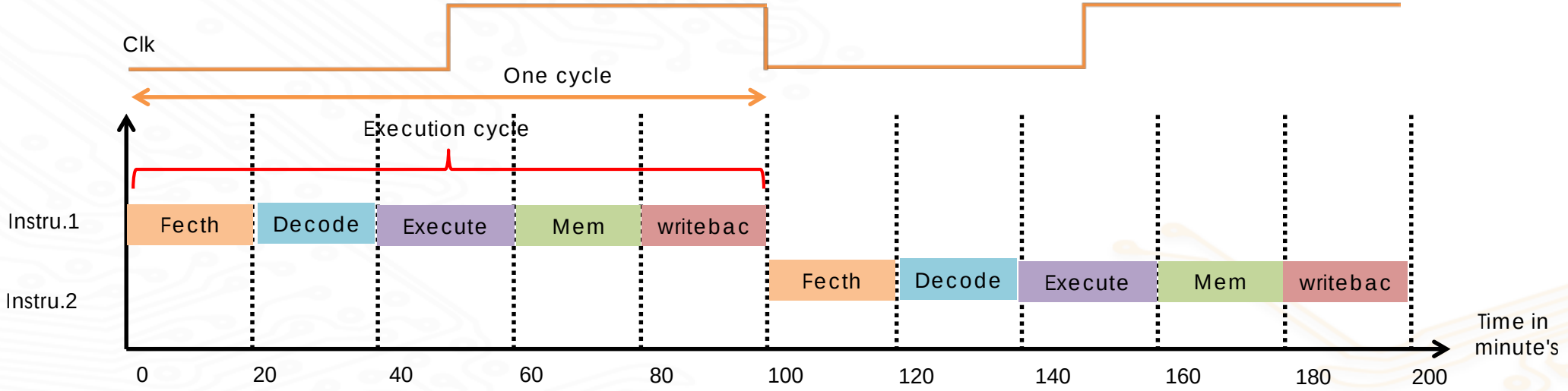
(Module 4: Instruction Level Parallelism(ILP))

Dr Smitha K. G.
School of Computer Science
And Engineering

Pre video summary

- Need of pipelining
- Pipeline data-path
- Challenges in ILP realization
- Data-dependence
- Pipeline hazards

Single cycle execution of instructions



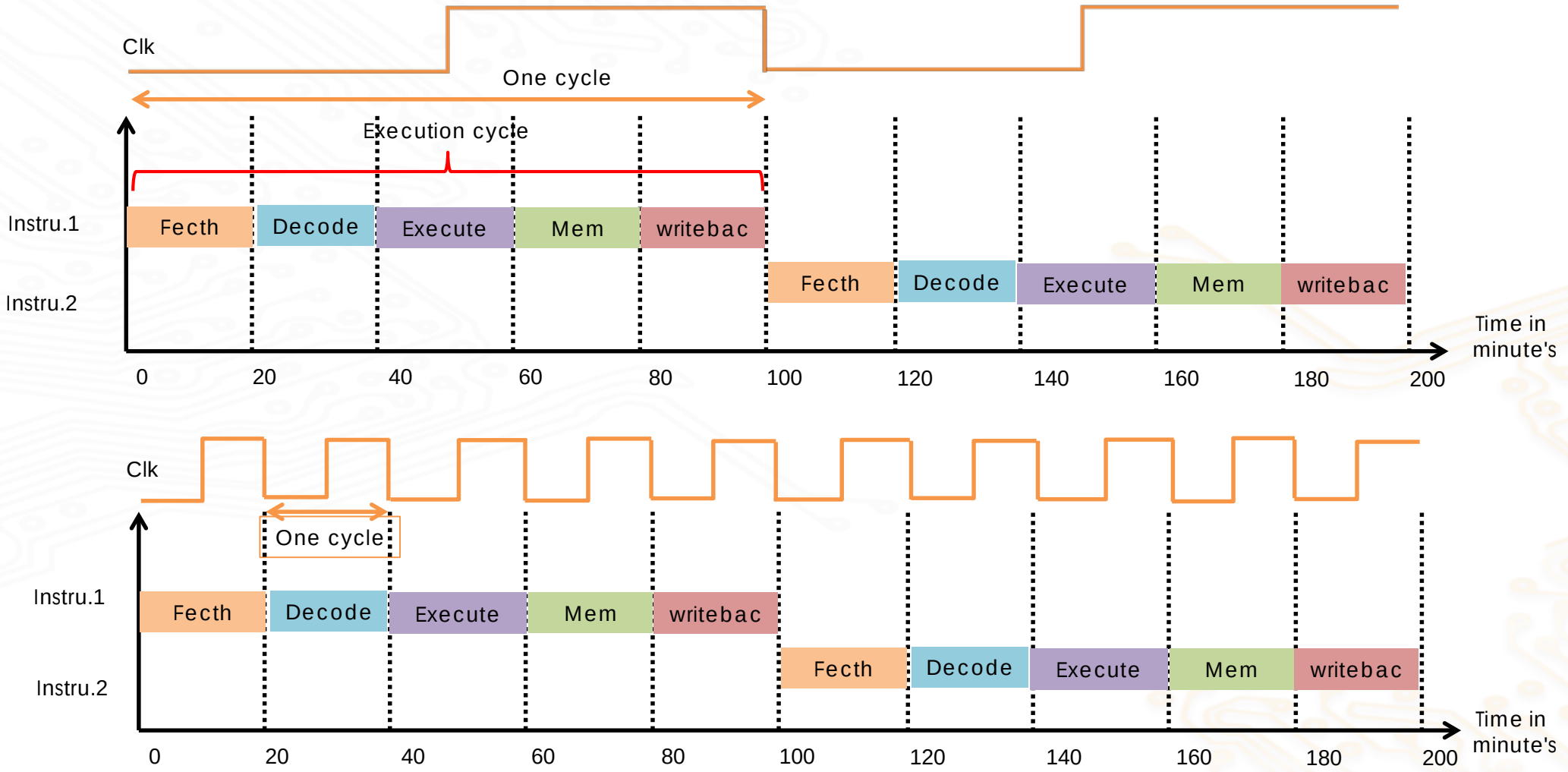
Exec time= IC x CPI x Time period

Time period is dependent on the worst case instruction in data path

How can we reduce execution time to improve performance?

Advantage: CPI =1 , no dependancy between instruction
Disadvantage: long clock period

Idea of multi-cycle- to reduce the clock period

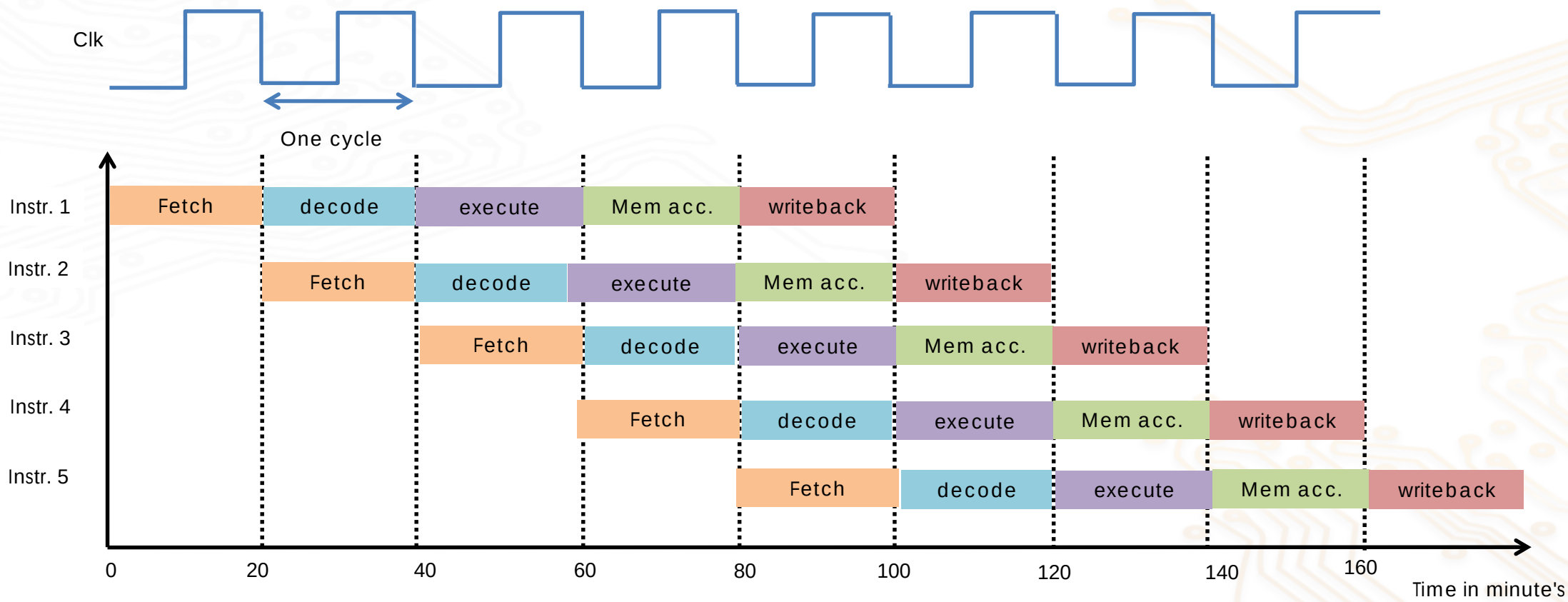


Advantage: Short clock period
Disadvantage: High CPI

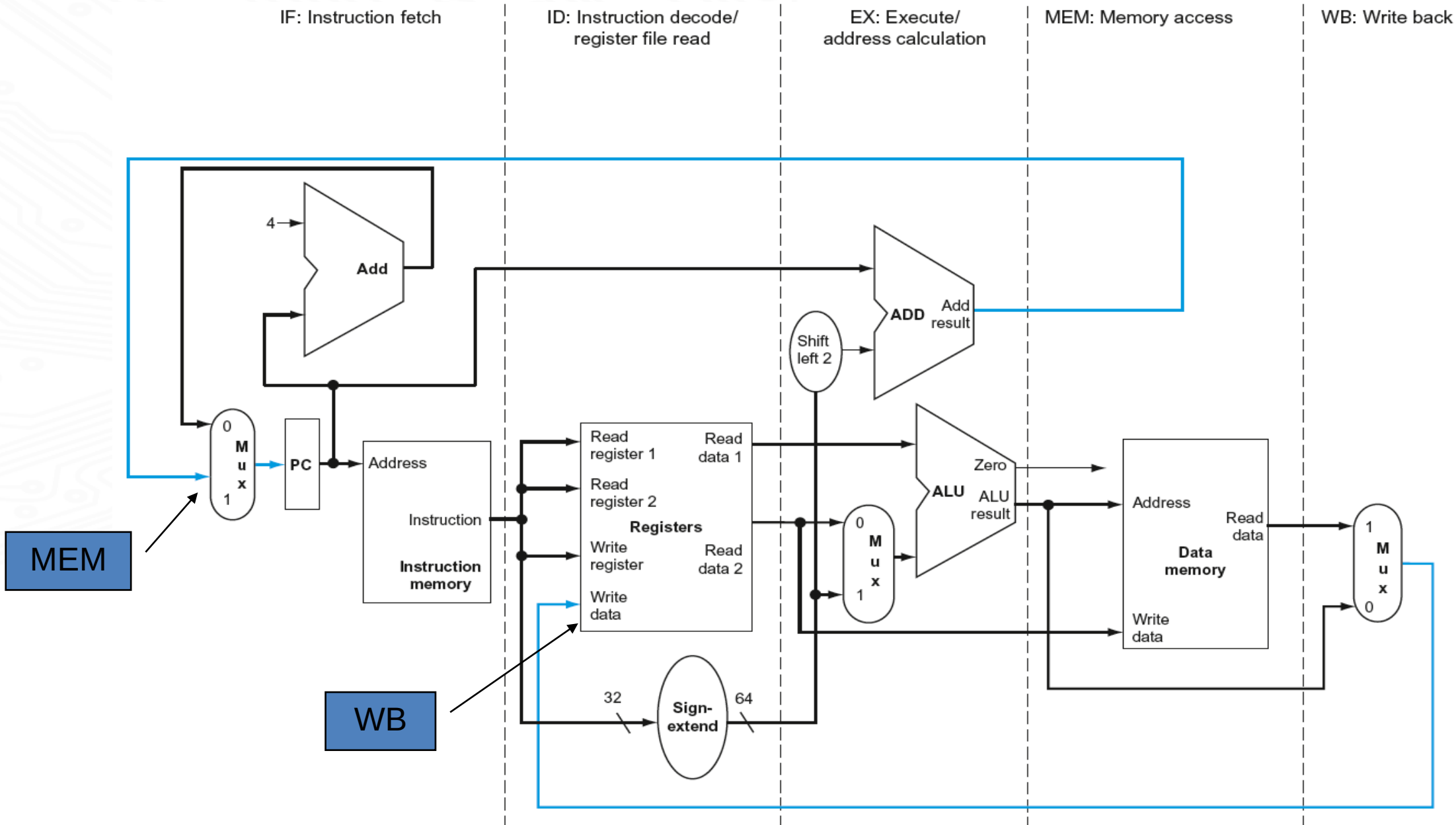
Pipeline datapath

So there is a need to keep the CPI as low as 1 and along with that we need to decrease the clock period.

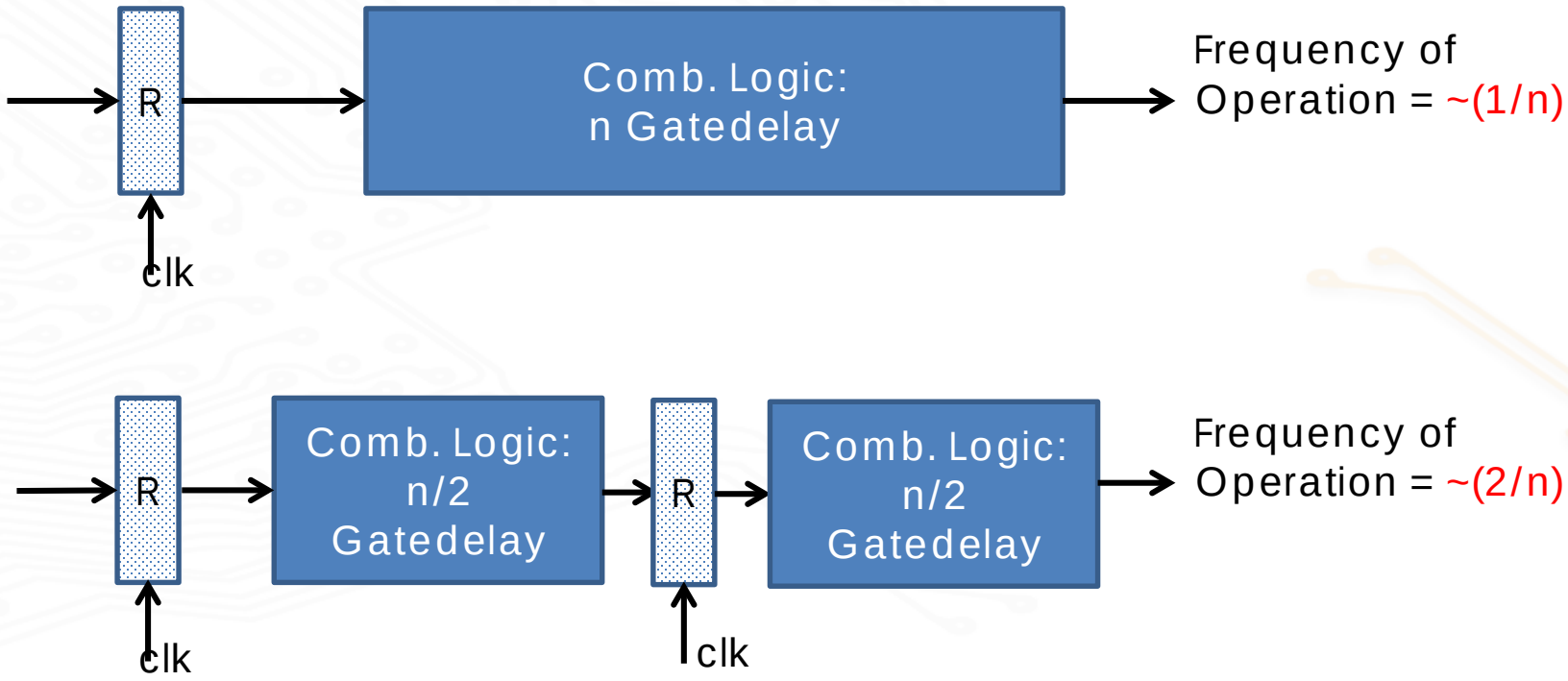
- **Pipelining** allows **multiple** sub-tasks to be carried out **simultaneously using independent resources**



Pipeline datapath



Ideal Pipelining



- So an n-stage pipeline means that the CPU can operate at a maximum of n times faster than without a pipeline (ideally!).
- There is an increase in latency due to register delays.

1. A single-cycle processor takes 4 ns to work on one instruction. We were able to convert the circuit into 4 stage pipeline. The stages have the following lengths: 1.2ns; 0.8ns; 0.7ns; 1.3ns; Please note that it takes 0.3 ns to latch its results into pipeline registers. Answer the following, assuming that there are no stalls in the pipeline.

   a) What is the minimum clock period for single-cycle and pipelined processors?

Answer:

- Minimum clock period for single cycle processor =
- Minimum clock period for pipelined processor=

b) If I was able to build a magical 1000-stage pipeline, where each stage took an equal amount of time, what speedup would I get?

Answer:

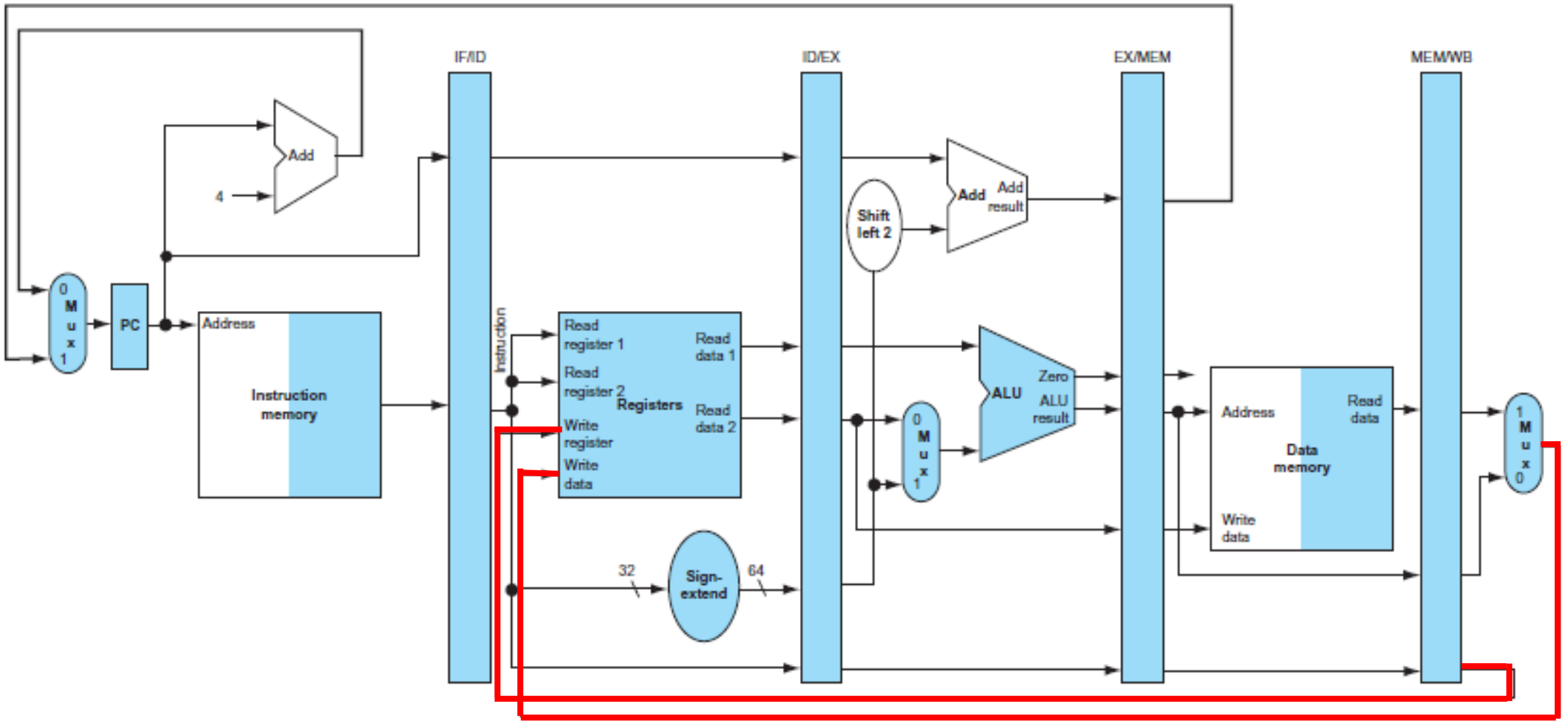
For a magical 1000 stage pipeline the speed up won't increase linearly with 1000 as the latching time is still present as a bottle neck.

The minimum time or length of one stage of a 1000 stage pipeline if we divide the work equally between stages=

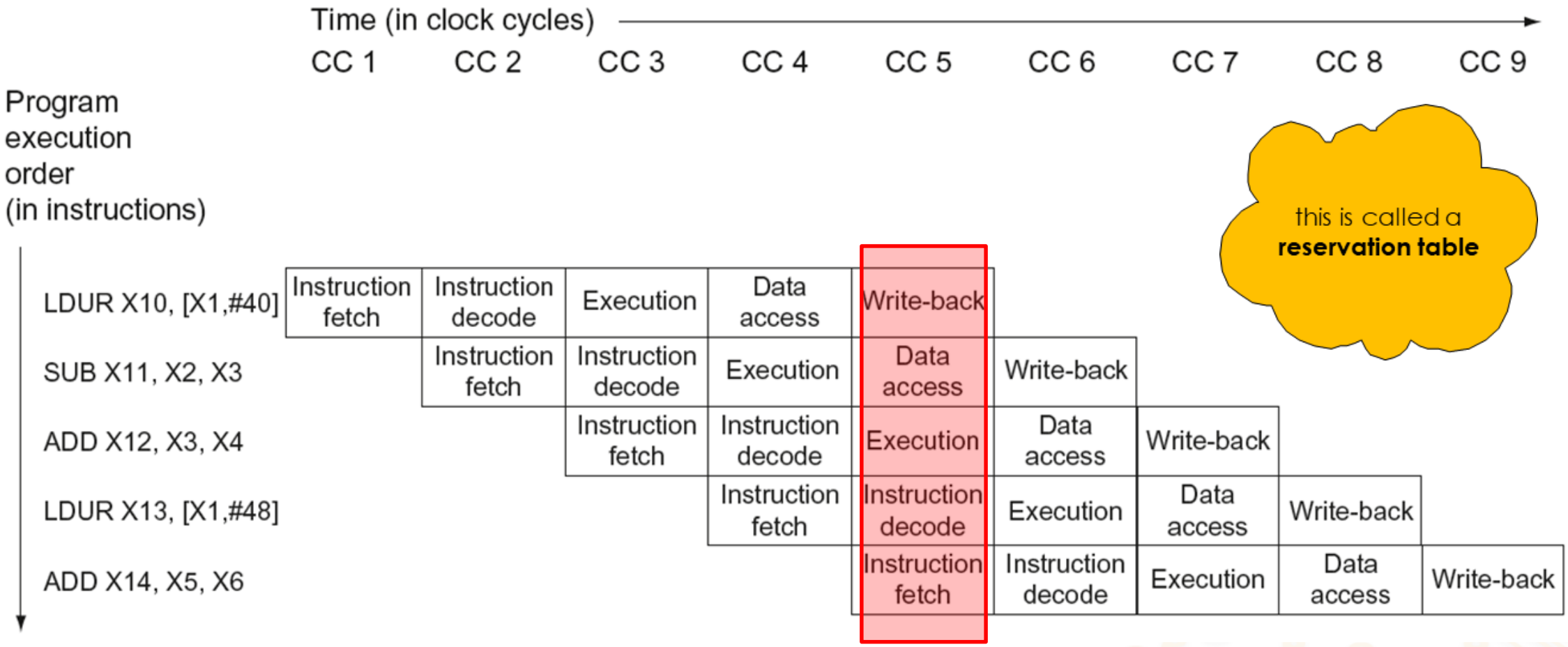
Cycle time for 1000 stage pipelined processor= Longest stage +latch=

Speed up provided by 1000-stage pipelined processor= execution time on un-pipelined/execution time in 1000 stage pipelined processor=

Pipelined datapath



Example of ideal pipelining

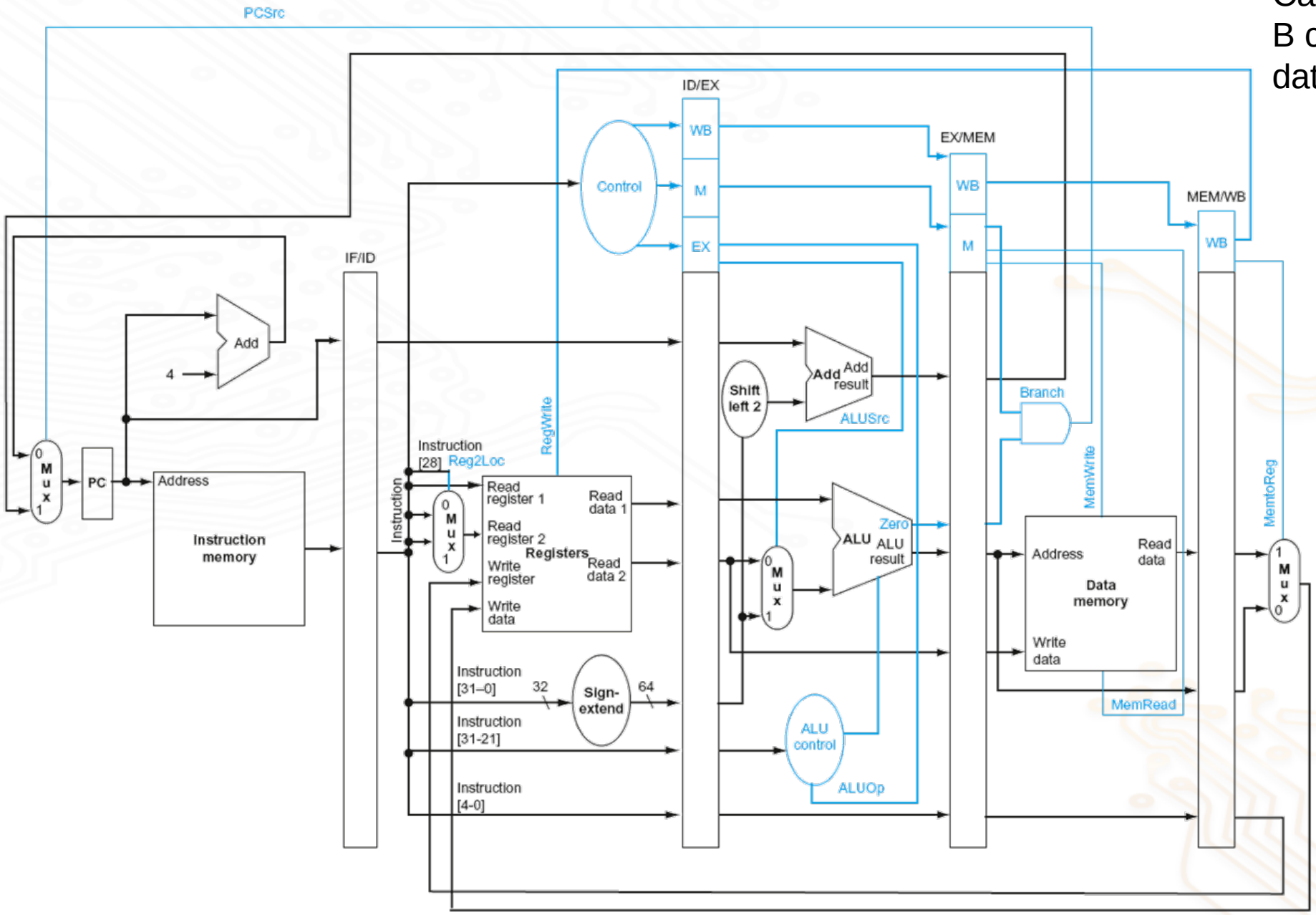


Each row of **reservation table** corresponds to a **pipeline** stage and each column represents a clock cycle.

Why this is ideal?

Pipelined datapath and control

Can you let me know if B can be done in this data path?



Challenges in instruction pipelining realisation

Ideal pipeline

- Identical task of all instructions
- Uniform decomposition of task
- Independent computations

So is pipeline always moving?

Real pipeline

- Except load instruction all other instructions do not need all stages
- Not all pipeline stages involve the same time to complete their respective sub-tasks
- The execution of one instruction may depend on one of the preceding instructions

Complications

- Datapath
  - **Many instructions in flight**
- Control
  - **Must correspond to multiple instructions**
- Instructions may have
  - **data and control flow *dependences***
  - **One may have to stall and wait for another**

Pipeline hazards and methods to solve

- **Structural hazard**
  - Can be solved by additional hardware elements
- **Data hazard**
  - Need to know the dependencies between data in the program and eliminate them
- **Control Hazard**
  - Need to predict the location of conditional /unconditional branch to avoid stalling

Data Dependencies

Types of data dependences

- True/flow dependence (**RAW** – read after write) – *j* cannot execute until *i* produces its result

  (*i* and *j* are two different instructions)

- Output dependence (**WAW** – write after write) – *j* cannot write its result until *i* has written its result

- Anti dependence (**WAR** – write after read) – *j* cannot write its result until *i* has read its sources

Data Dependencies (examples)

```
I1:   LDUR    X3, [X2, #0];
I2:   AND     X5, X3, X4;
```

```
I1:   ADD     X0, X2, X1;
I2:   LDUR    X0, [X3, #0];
```

```
I1:   ADD     X0, X2, X1;
I2:   STUR    X0, [X3, #0];
```

```
I1:   ADD     X0, X2, X1;
I2:   CBZ     X0, #2;
```