Different ways to speed up computer systems

Keivan Navi

Winter 2025

Outline

- Pipeline
- Super-Pipeline
- Super-Scalar

General Performance Metrics

- Performance can be determined by:
  - Instruction Count
  - Clock Rate
  - CPI (Clocks Per Instructions)
- Execution Time = Instruction Count * CPI * Cycle Time
- The Instruction Count is governed by Compiler's Techniques and Architectural decisions.
- The CPI is primarily governed by Architectural decisions.
- The Cycle Time is governed by technology improvements and Architectural decisions.

Unpipelined Processor

- Only ONE instruction can be resident at the processor at any given time.
- The whole processor is considered as ONE stage, k=1.
  CPI = 1 / IPC

One Instruction resident in Processor

The number of stages K=1

First method: Pipelining

Pipelining

- Pipelining is a key implementation technique used to build fast processors.
- It allows the execution of multiple instructions to overlap in time.
- The throughput of an instruction pipeline is the measure of how often an instruction exits the pipeline.

So :

Pipelining is a way of speeding up execution of instructions by overlapping the execution of multiple instructions.

Pipelining

- Pipelining increases throughput, but not latency.

Limitations:

- Computations must be divisible into stage size
- Pipeline registers add overhead

Pipelining Processors

- Two possible implementations of the MIPS architecture:
  - Single-cycle datapath (It executes each instruction in just one clock cycle, but the cycle time may be very long.)
  - Multicycle datapath (It has much shorter cycle times, but each instruction requires many cycles to be executed.)

- Pipelining gives the best of both worlds and is used in almost every modern processor.
  - Cycle times are short so clock rates are high.
  - But we can still execute an instruction in about one clockcycle!

Single Cycle Datapath	CPI = 1	Long Cycle Time
Multi-cycle Datapath	CPI = ~4	Short Cycle Time
Pipelined Datapath	CPI = ~1	Short Cycle Time

Pipelining Processors

- Executing a MIPS instruction can take up to five steps:

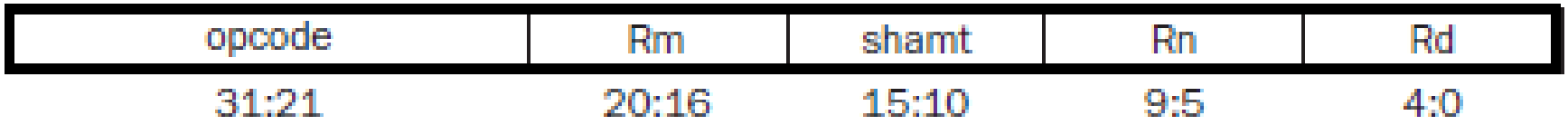
Step	Name	Description
Instruction Fetch	IF	Read an instruction from memory.
Instruction Decode	ID	Read source registers and generate control signals.
Execute	EX	Compute an R-type result or a branch outcome.
Memory	MEM	Read or write the data memory.
Writeback	WB	Store a result in the destination register.

- However not all instructions need all five steps.

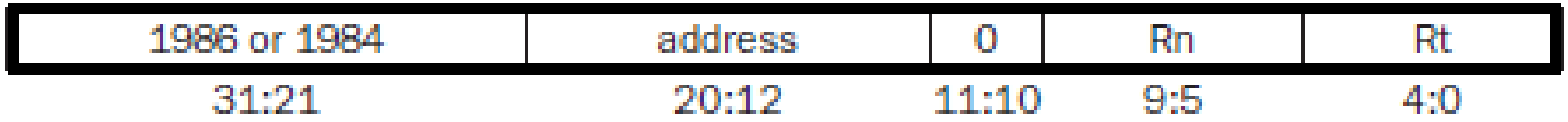
Instruction	Steps required
beq	IF ID EX
R-type	IF ID EX WB
sw	IF ID EX MEM
lw	IF ID EX MEM WB

Types of Instructions

- There are 3 different types of instructions:
- 1- Register Type (R-Type):

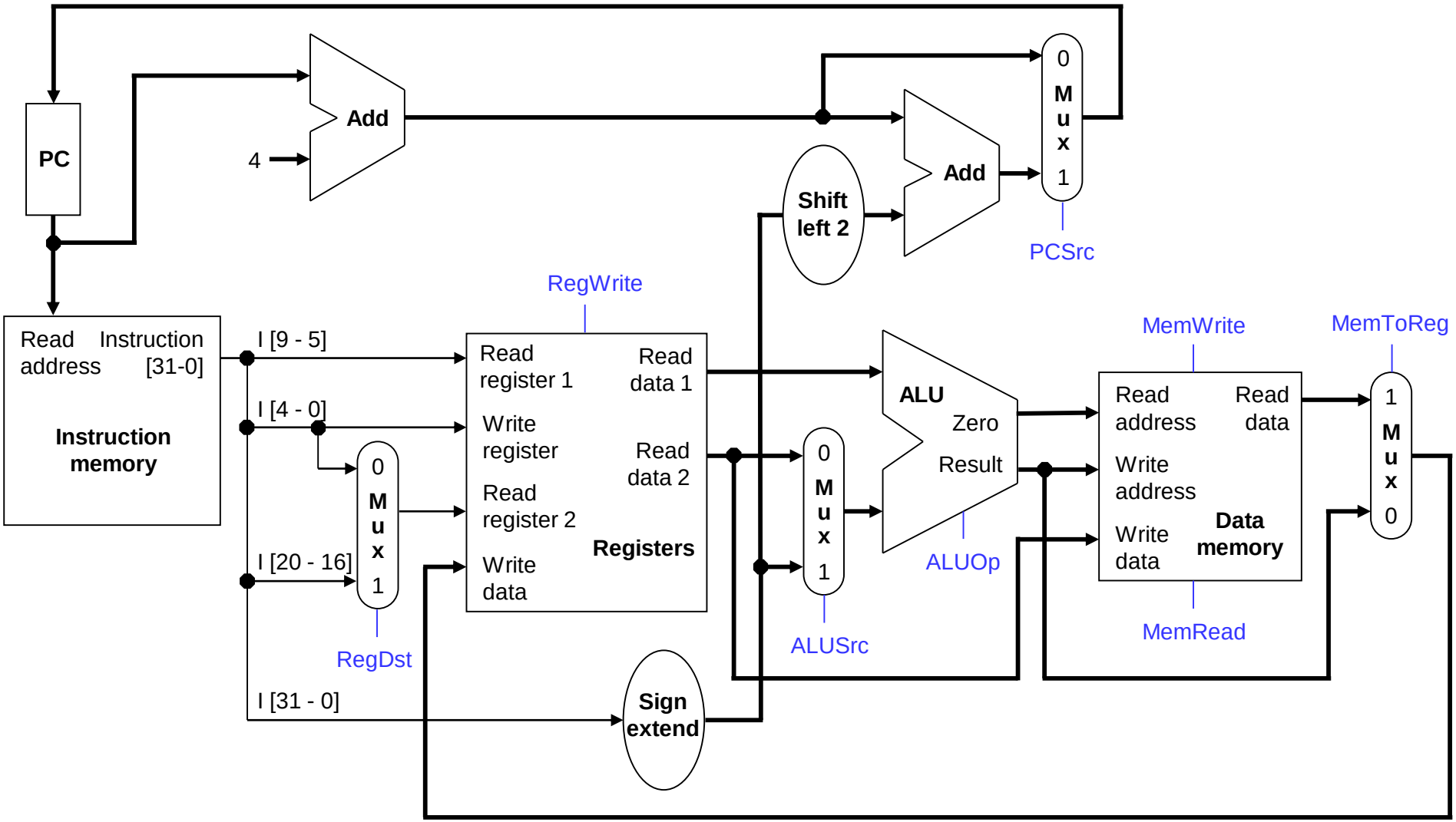


- 2-Laod Store:



- 3-Branch:

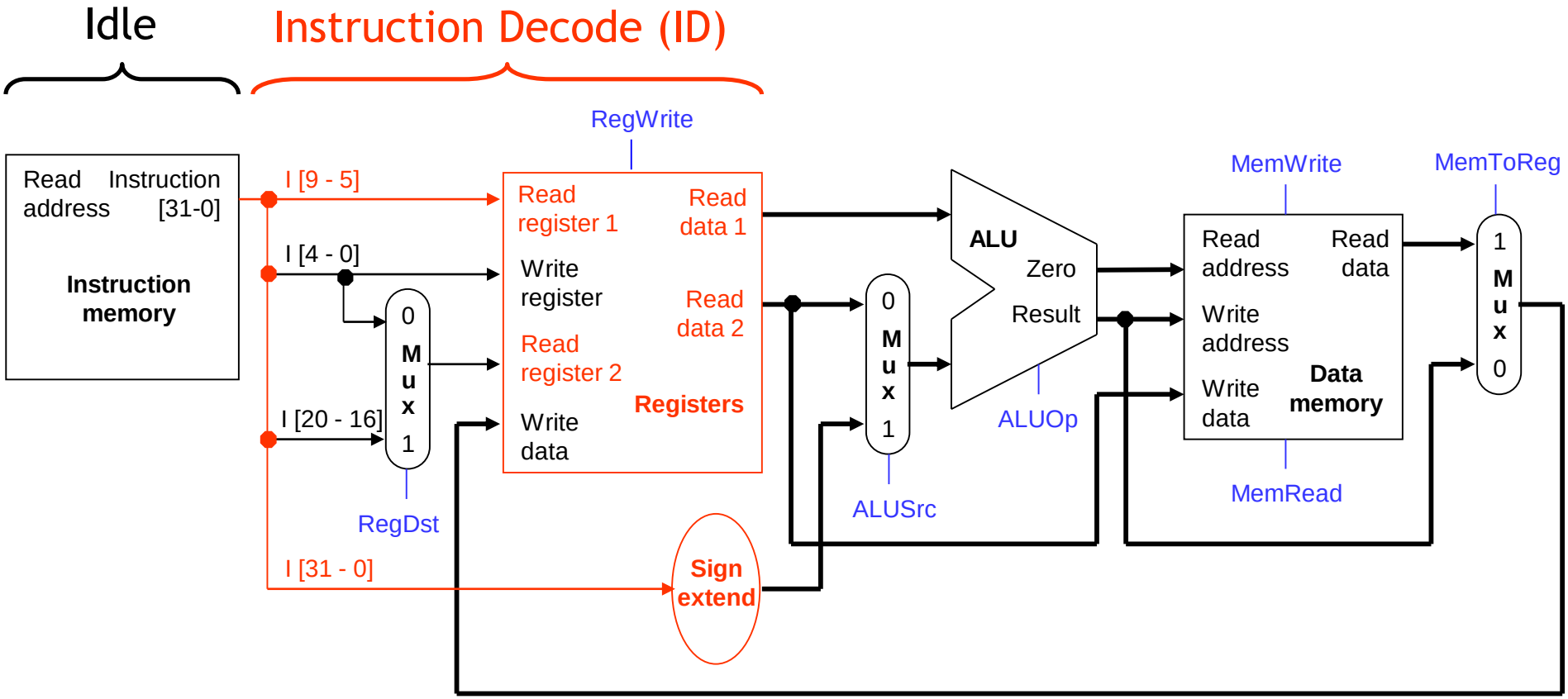
Single-cycle datapath diagram



- How long does it take to execute each instruction?

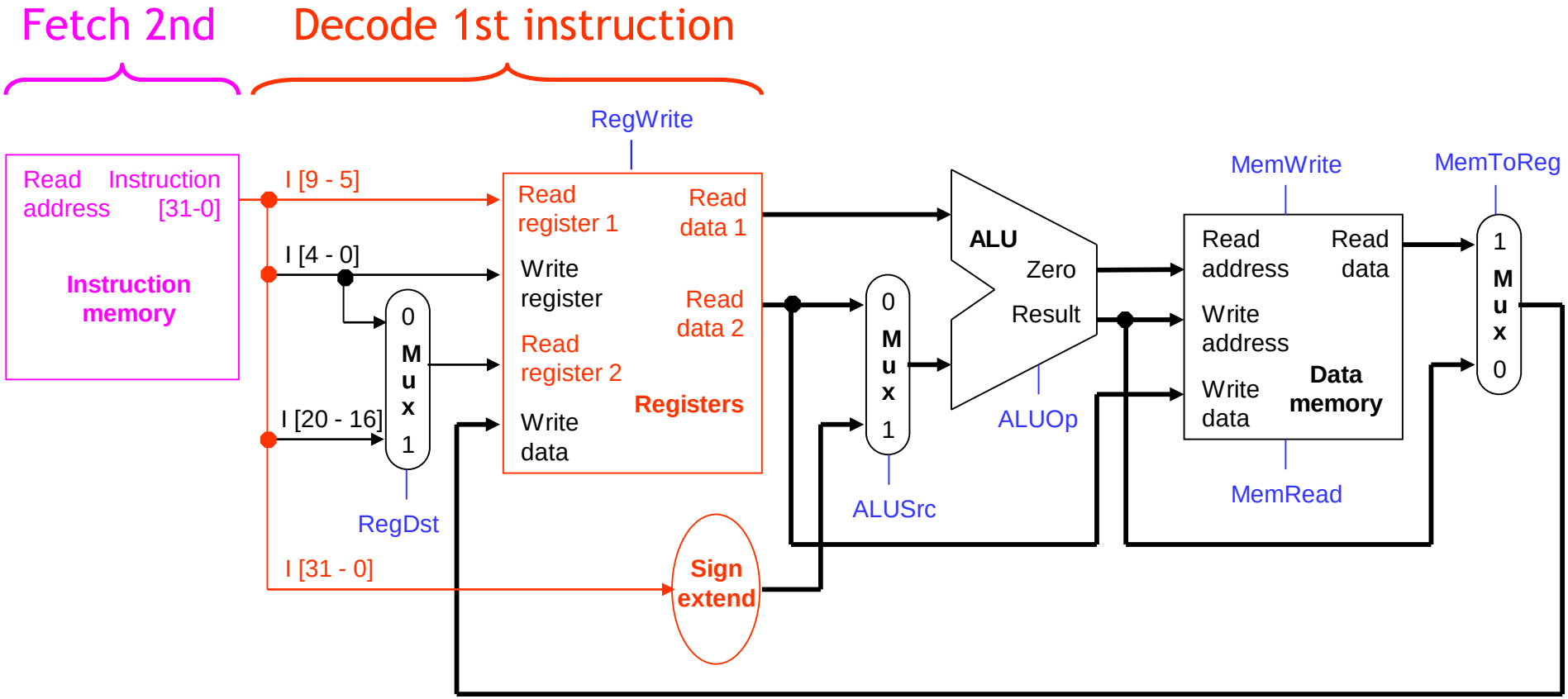
Pipelining

We shouldn't have to wait for the entire instruction to complete before we can reuse the functional units. For example, the instruction memory is free in the Instruction Decode step as shown below, so...



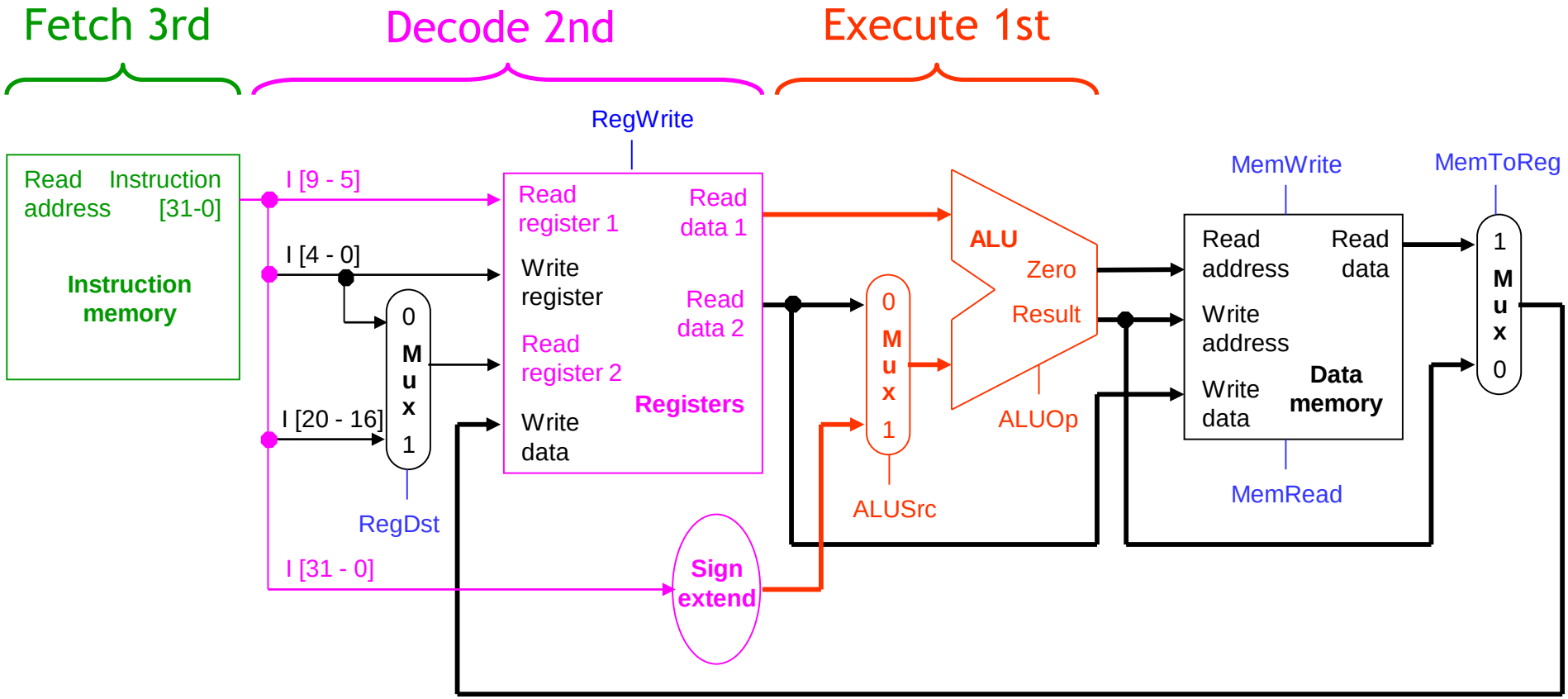
Decoding and fetching together

Why don't we go ahead and fetch the next instruction while we're decoding the first one?



Executing, decoding and fetching

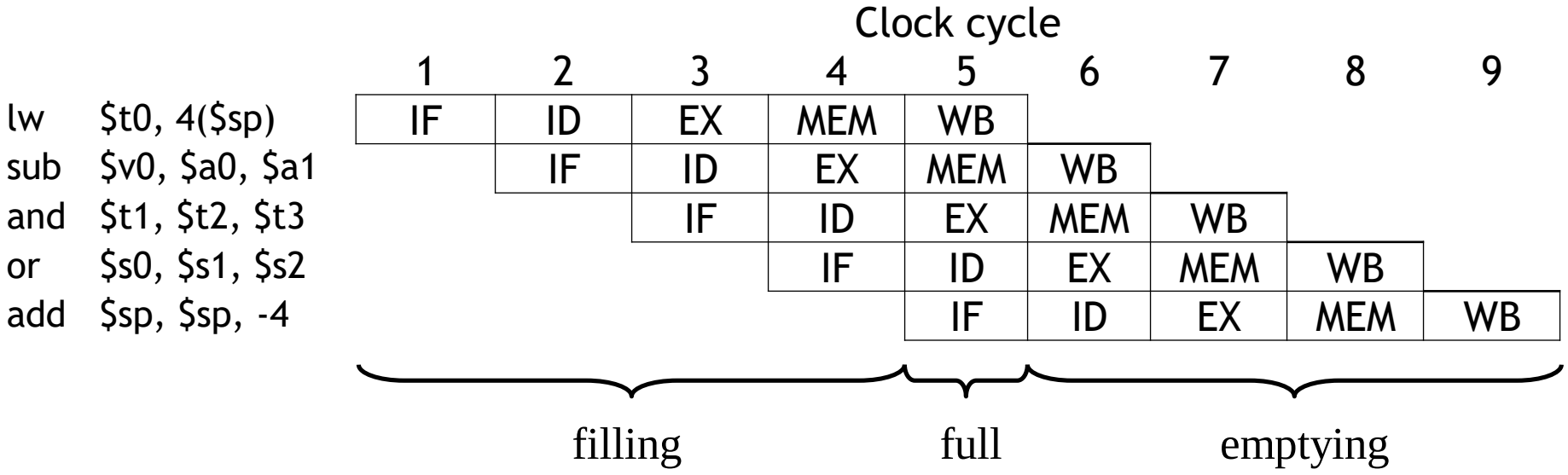
Similarly, once the first instruction enters its Execute stage, we can go ahead and decode the second instruction. But now the instruction memory is free again, so we can fetch the third instruction!



Making Pipelining Work

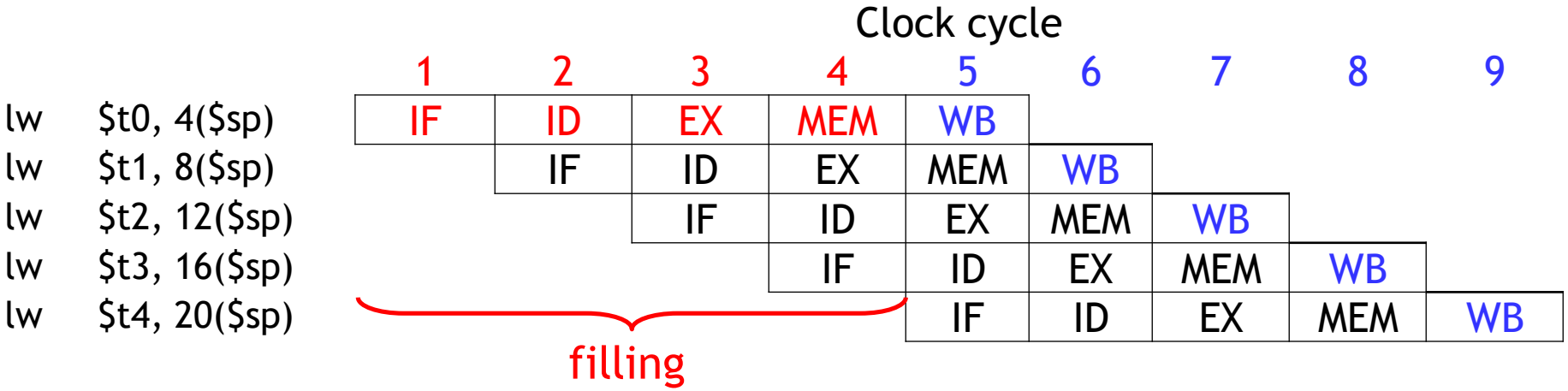
- We'll make our pipeline 5 stages long, to handle load instructions as they were handled in the multi-cycle implementation.
    - Stages are: IF, ID, EX, MEM, and WB
- We want to support executing 5 instructions simultaneously: one in each stage.

Pipeline terminology



- The pipeline depth is the number of stages—in this case, five.
- In the first four cycles here, the pipeline is filling, since there are unused functional units.
- In cycle 5, the pipeline is full. Five instructions are being executed simultaneously, so all hardware units are in use.
- In cycles 6-9, the pipeline is emptying.

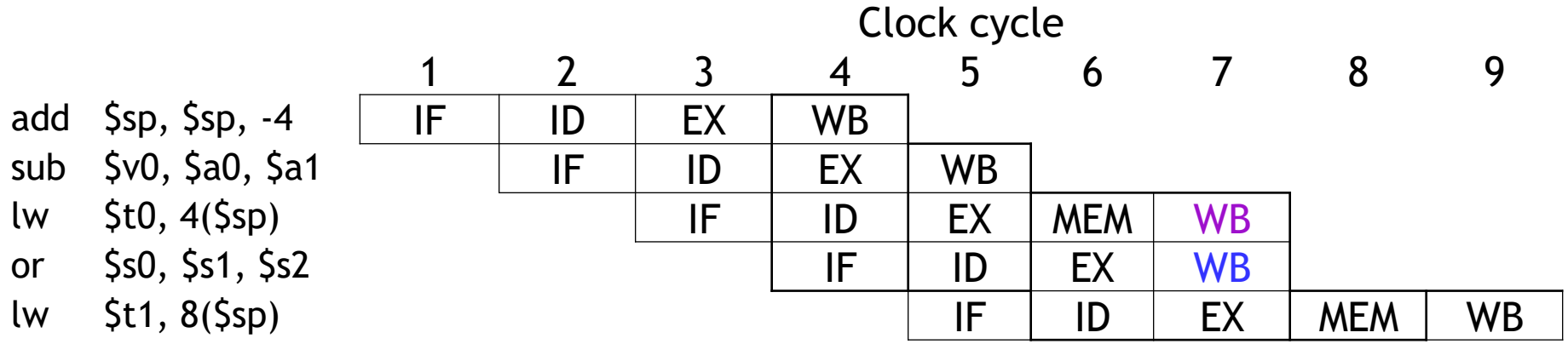
Pipeline terminology



- Execution time on ideal pipeline:
  - time to fill the pipeline + one cycle per instruction
  - What is the execution time for N instructions?
- Compare with other implementations:
  - Single Cycle: (8ns clock period)
- How much faster is pipelining for N=1000 ?

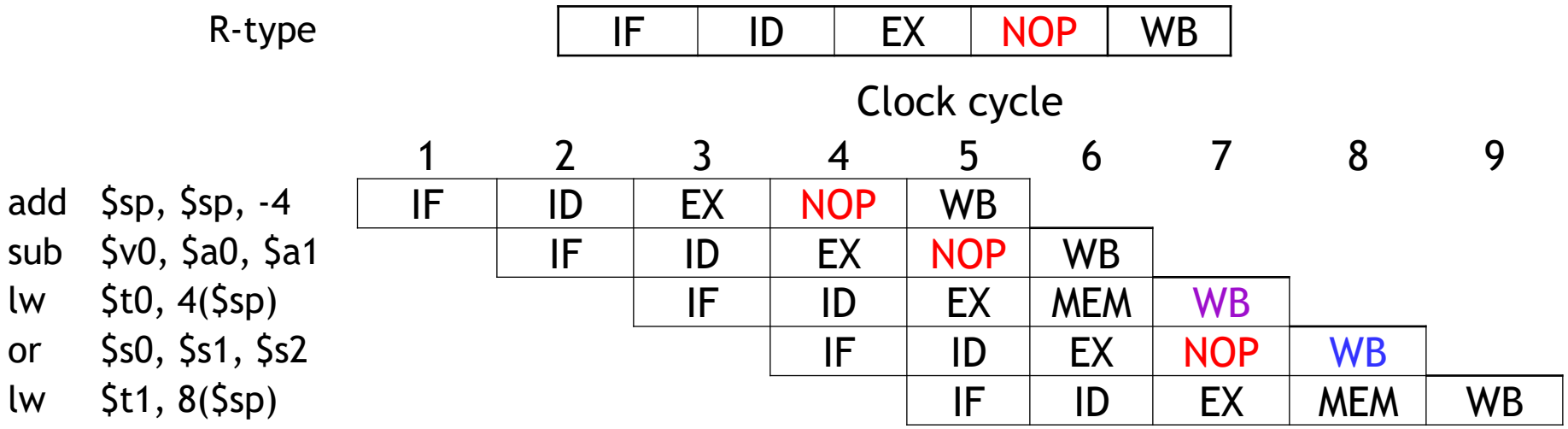
Important Observation

- Each functional unit can only be used once per instruction.
- Each functional unit must be used at the same stage for all instructions. See the problem if:
  - Load uses Register File's Write Port during its 5th stage
  - R-type uses Register File's Write Port during its 4th stage

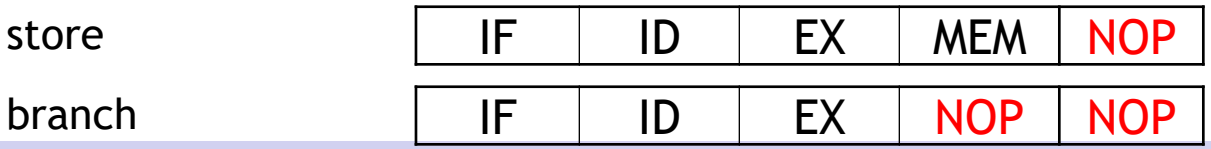


A solution: Insert NOP stages

- Enforce uniformity
  - Make all instructions take 5 cycles.
  - Make them have the same stages, in the same order
    - Some stages will do nothing for some instructions

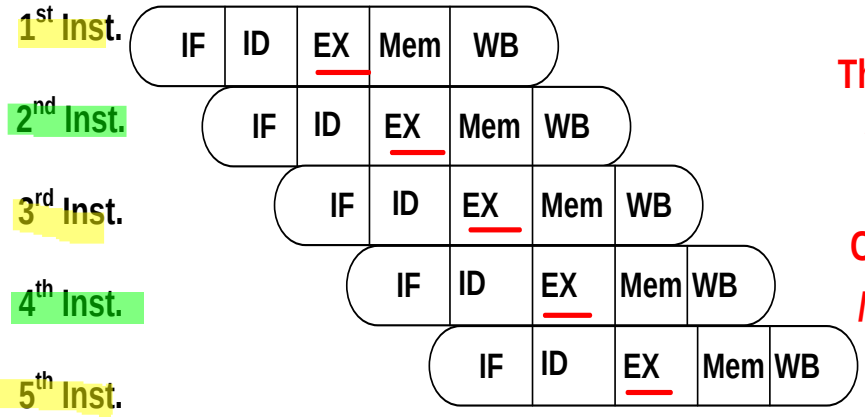


- Stores and Branches have NOP stages too…



Therefore…

- K –number of pipe stages, instructions are resident at the processor at any given time.
- In our example, K=5 stages, number of parallelism (concurrent instruction in the processor) is also equal to 5.
- One instruction will be accomplished each clock cycle, CPI = IPC = 1

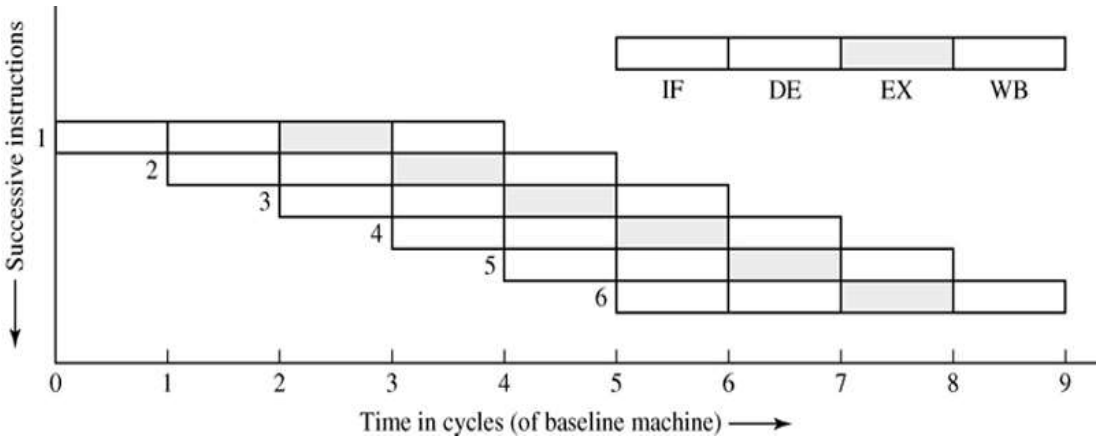


The number of stages K=5

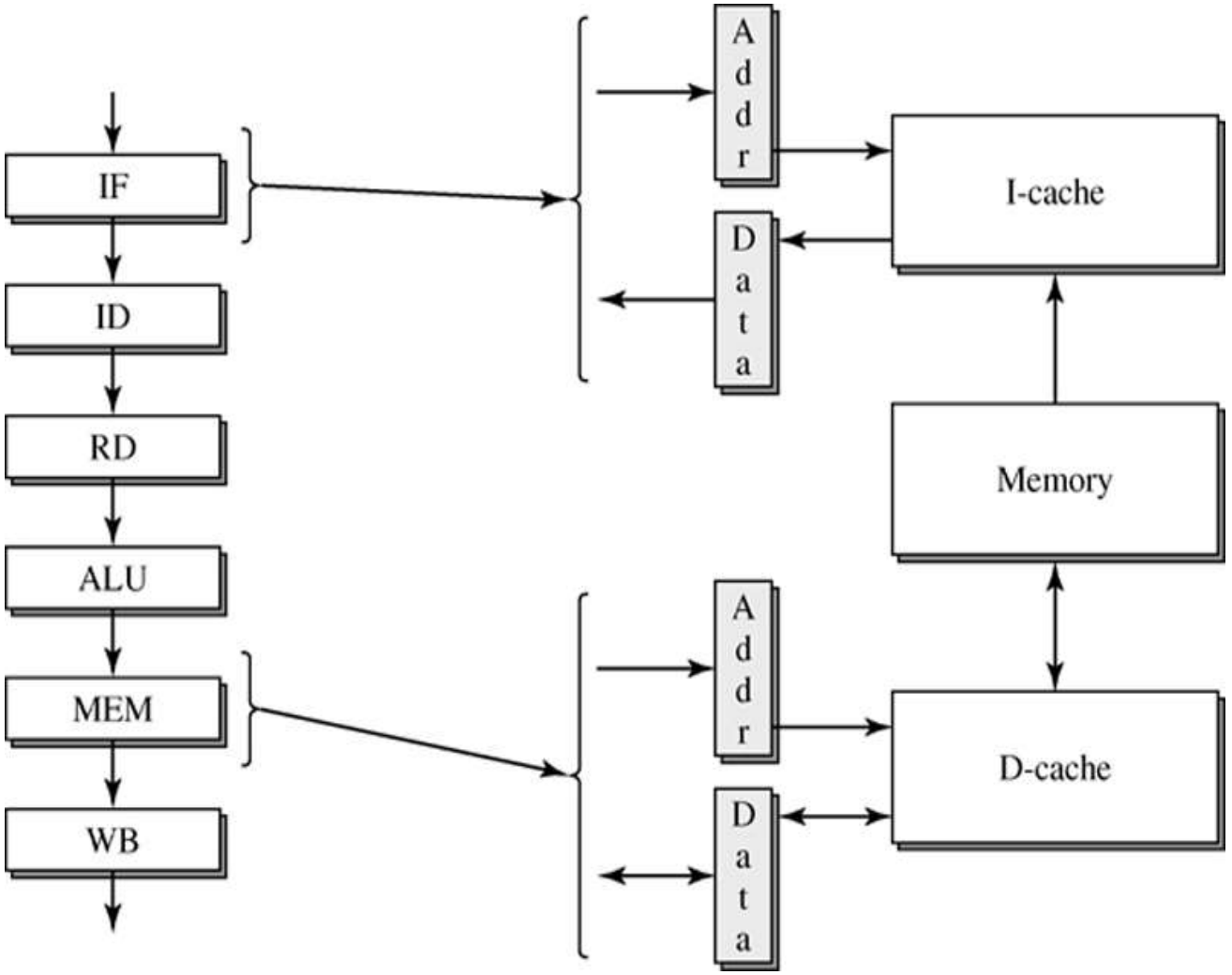
Ideally,

CPI = IPC = 1

//'ims = k =5



Example of a Six-Stage Pipelined Processor



Pipelining & Performance

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

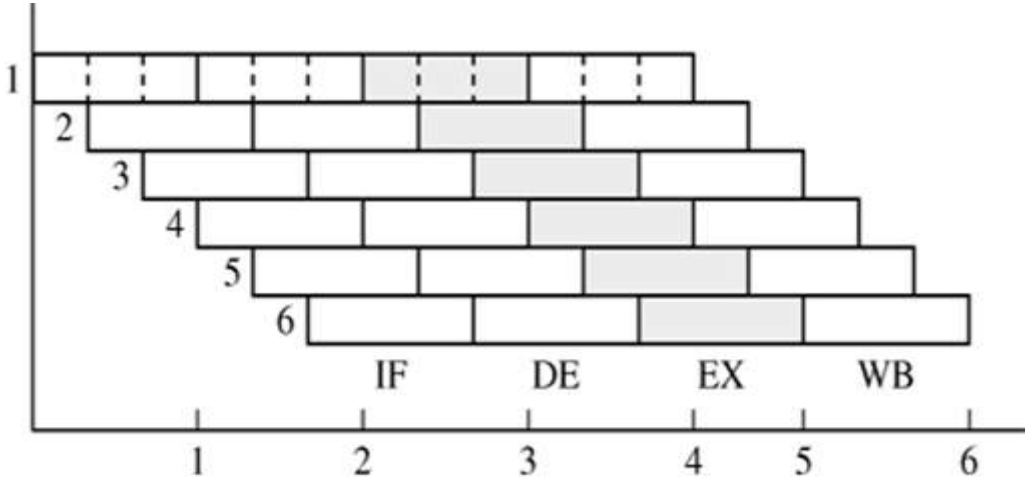
- The Pipeline Depth is the number of stages implemented in the processor, it is an architectural decision, also it is directly related to the technology. In the previous example K=5.
- The Stall's CPI are directly related to the code's instructions and the density of existing dependences and branches.
- Ideally the CPI is ONE.

Second method: Super-Pipelining

Super-Pipelining Processor

- IF (Instruction Fetch) : requires a cache access in order to get the next instruction to be delivered to the pipeline.
- ID (Instruction Decode) : requires the control unit to decode the instruction to identifying what it does.
- Obviously the fetch stage cache access requires much more time than the decode stage –combinational logic.
- We can subdivide each pipe-stage into m other stages of smaller amount of time required for each stage: minor cycle time.

Super-Pipelining Processor

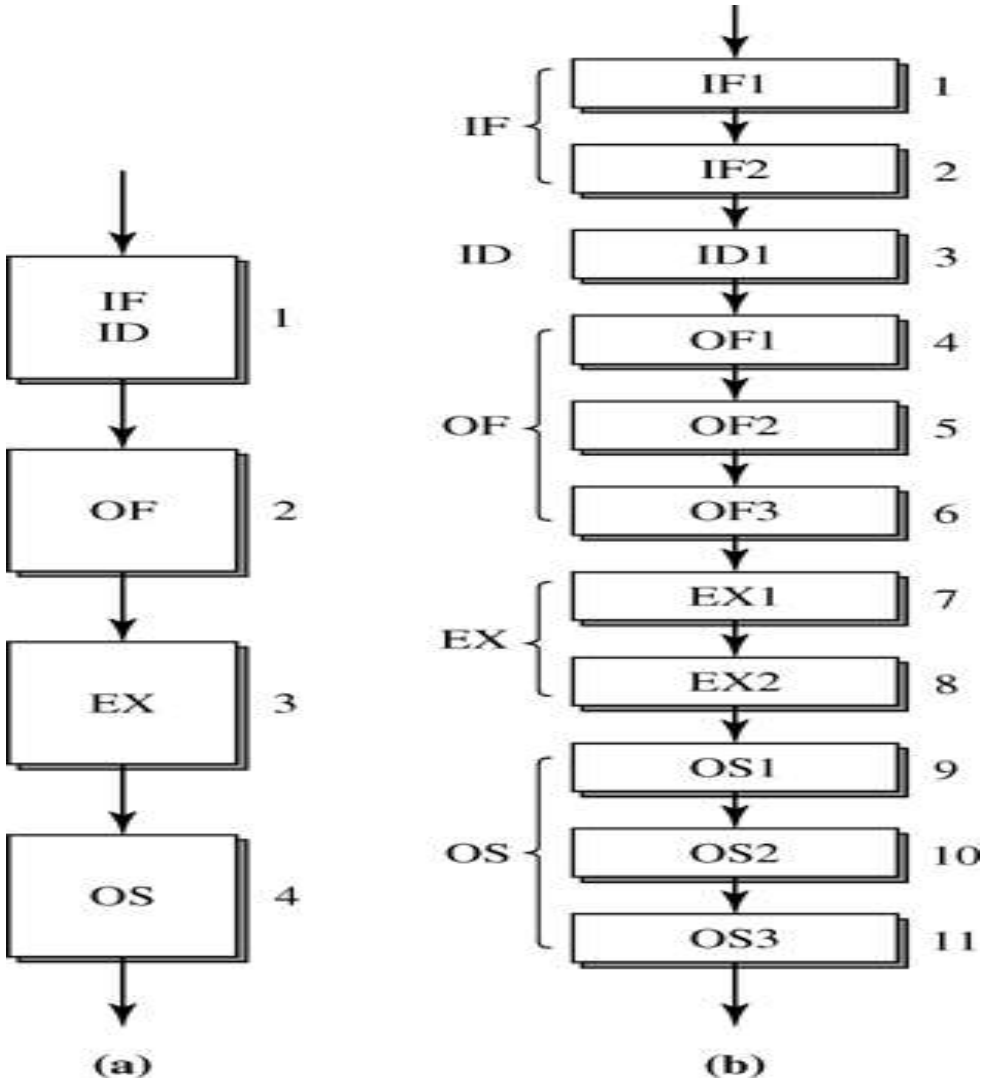


Super-pipelined machine of Degree m=3

- The machine can issue a new instruction every minor cycle.
- Parallelism = K * m.
- For this figure:

Parallelism = K * m = 4 * 3 = 12

Stage Quantization



(a)four-stage instruction pipeline, (b)eleven-stage instruction pipeline

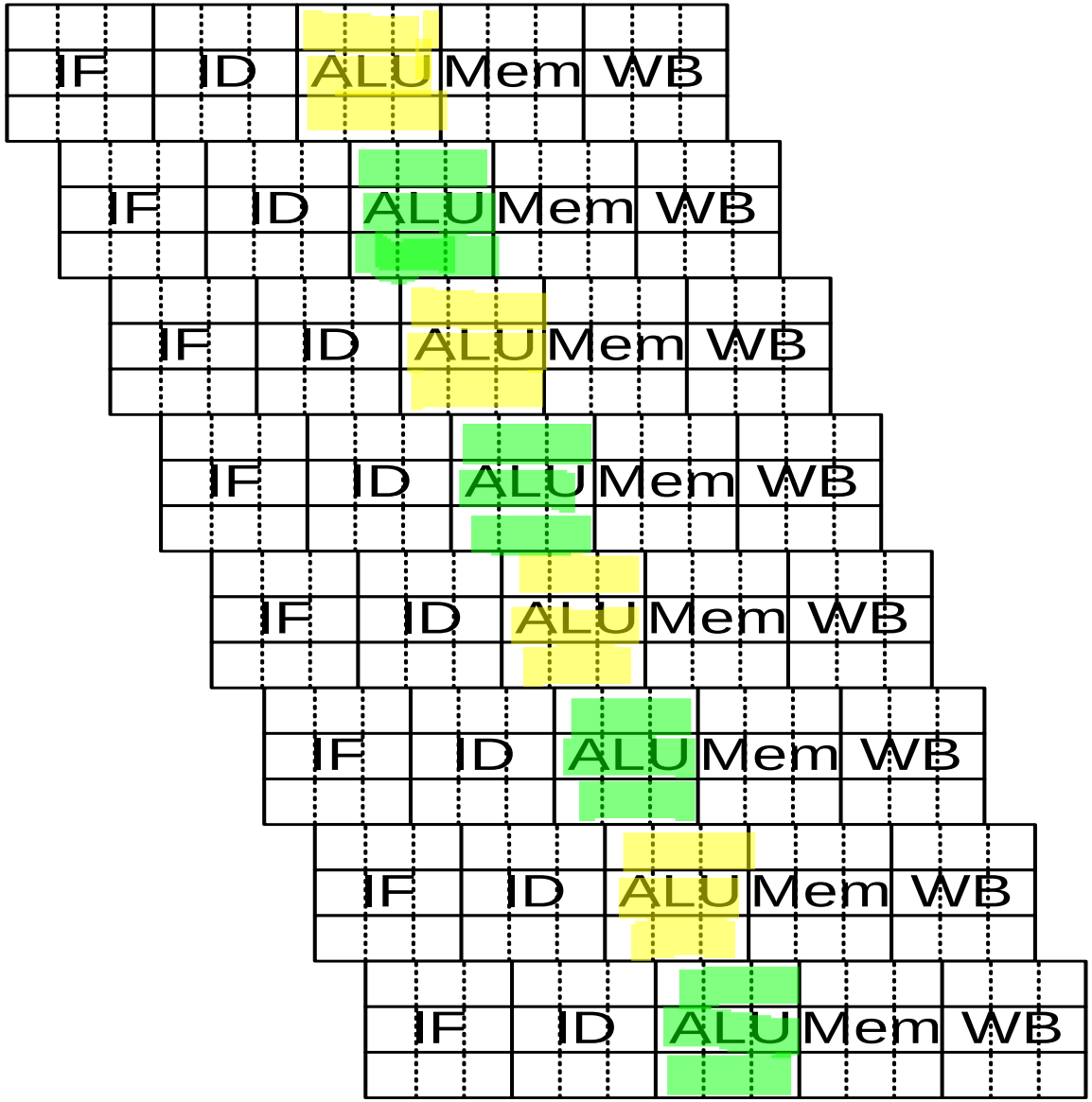
Third method: Superscaling

Characteristics of Superscalar Machines

- Simultaneously advance multiple instructions through the pipeline stages.
- Multiple functional units => higher instruction execution throughput.
- Able to execute instructions in an order different from that specified by the original program.
- Out of program order execution allows more parallel processing of instructions.

Superpipelined Superscalar Machine

- The superscalar degree is determined by the issue parallelism n, the sub-stages m and the number of stages k.
- Parallelism = K * m * n
- For this figure:
  Parallelism = K * m * n
  = 5 * 3 * 3 = 45



Deeper Pipeline, a Solution?

- As mentioned before, the performance is proportional to Instruction Count, Clock Rate and IPC.
- Deeper pipeline has fewer logic gate levels in each stage, this leads to a shorter cycle time and higher clock rate.
- But deeper pipeline can potentially cause higher penalties for dealing with inter-instruction dependences.

Bounded Pipelines Performance

- Scalar pipeline can only initiate at most one instruction every cycle, hence IPC is fundamentally bounded by ONE.
- To get more instruction throughput, we must initiate more than one instruction every machine cycle.
- Hence, having more than one instruction resident in each pipeline stage is necessary: parallel pipeline.

From Scalar to Superscalar Pipelines

- Superscalar pipelines are parallel pipelines: able to initiate the processing of multiple instructions in every machine cycle.
- Superscalars are diversified; they employ multiple and heterogeneous functional units in their execution stages.
- They are implemented as dynamic pipelines in order to achieve the best possible performance without requiring reordering of instructions by the compiler.

Acknowledgement

- I would like to thank my dear PhD student Samaneh Firouzi for her help, enthusiasm and support for designing this power point.